Títol: Sistema multi-projector per a realitat virtual

Volum: 1/1
Alumne: Rodrigo Pizarro Lozano

Director/Ponent: Pere Brunet Crosa
Departament: Llenguatges i Sistemes Informàtics
Data: Gener 2009

DADES DEL PROJECTE

Títol del projecte:	Sistema multi-projector per a realitat virtual
Nom de l'estudiant:	Rodrigo Pizarro Lozano
Titulació:	Enginyeria Superior en Informàtica
Crèdits:	37,5
Director/Ponent:	Pere Brunet Crosa
Departament:	Llenguatges i Sistemes Informàtics

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President:	Isabel Navazo Alvaro
Vocal:	German Santos Boada
Secretari:	Pere Brunet Crosa

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Sistema multi-projector per a realitat virtual

Rodrigo Pizarro Lozano

Gener 2009

Índex

Índex de figures

Capítol 1

Introducció

Aquest projecte ha estat la culminació d'anys d'estudi, i alhora un gran repte per la seva complexitat. Aquesta magnitud ha estat suficient per abastar dos PFC. Cal tenir present aquest fet, perquè es faran sovint referències a la memòria del meu company.

En aquest primer capítol es pretén explicar *grosso modo* tant el tema que es tracta en el present projecte de final de carrera com l'organització en què es presenta la memòria.

1.1 Descripció del projecte

El projecte consisteix en dissenyar un sistema immersiu de realitat virtual totalment portable, construir-ne un prototipus i comprovar i validar el seu funcionament.

El prototipus consta de dues *unitats independents* amb un projector i una càmera digital cadascuna, necessaris per a realitzar el calibratge del sistema. A més a més, també inclou un mecanisme que permet fer el seguiment de l'usuari[1].

1.2 Motivació

Com ja hem dit, aquest projecte de fi de carrera està pensat per ser la primera part d'un projecte molt més ambiciós. L'objectiu final d'aquest és crear un sistema de realitat virtual anomenat *cave flexible*, molt semblant a un sistema cave. La motivació de crear aquesta *cave flexible* seria eliminar els principals inconvenients de la cave tradicional. L'explicació de què és aquesta *cave* i de quins són els problemes a resoldre està detallada en el capítol 2.3.1 i 3.2, respectivament.

El sistema *cave flexible* pretén solucionar aquests inconvenients de la següent manera: s'utilitzaran projectors petits, com els que podria tenir qualsevol a casa. Això provoca canvis importants que es descriuen a l'apartat 9.1

Parlant de motivacions comercials, una que pot tenir molt futur, és en el món dels videojocs, ja que els usuaris cada cop volen pantalles més grans, amb més resolució i, fins i tot, amb visió estèreo. Aquest projecte en concret, se centra precisament en l'àmbit de les aplicacions 3D, potenciant l'efecte immersiu que s'aconsegueix amb la utilització d'aquest tipus de visionat, i altres tècniques, com el *head tracking*.

Un àmbit radicalment diferent al dels videojocs on pot tenir molt interès és en la medicina, on es creu podria suposar una gran revolució. Actualment, les pantalles que s'utilitzen han de

[1] Aquest seguiment es realitza amb el comandament de la videoconsola de Nintendo *Nintendo Wii*

concentrar una quantitat d'informació molt gran en dispositius 2D força reduïts; afegir-hi una tercera dimensió i eliminant les restriccions de mida, la feina de molts especialistes pot veure's simplificada en gran mesura, atès que disposaran d'una eina molt més potent per a visualitzar el problema que afecti al pacient.

Altres escenaris són el món de la publicitat o diversos esdeveniment com ara concerts, esports o espectacles a l'aire lliure. En aquests llocs ja existeixen pantalles de gran mida, però el sistema que es proposa dóna una solució molt més barata que aquestes, oferint, a més a més, una qualitat d'imatge molt superior.

També cal emfatitzar que es tracta d'un sistema molt flexible i portable, el qual resultaria ideal per a esdeveniments temporals o itinerants, com per exemple gires de concerts o presentacions arreu del món. El seu transport i manteniment resultaria molt senzill, i els mòduls que composen la solució final simplificarien molt el muntatge de la infraestructura, ja que cap d'ells seria indispensable i podrien ser reemplaçats amb facilitat.

1.3 Objectius

Tal i com ja s'ha introduït a l'apartat 1.1, el principal objectiu del projecte és desenvolupar un *sistema immersiu de realitat virtual*. A continuació es descriuen una sèrie de característiques que el fan singular:

Portabilitat El sistema ha de poder utilitzar-se en diferents ubicacions i no ha d'estar, per tant, lligat a un emplaçament fix.

Calibratge automàtic La portabilitat del sistema requereix que es pugui adaptar fàcilment a cadascun dels diferents entorns on s'utilitzi. Per tal de cobrir aquesta necessitat de la forma més còmoda possible, es vol que sigui capaç de configurar-se automàticament[2].

Head tracking Per tal d'assolir un nivell de realisme més elevat, el sistema implementa un mecanisme que fa un seguiment de la posició de l'usuari, de tal manera que el que s'hi mostra sigui coherent amb la seva ubicació real relativa a la pantalla.

Aplicació d'usuari Es necessita una aplicació que utilitzi tota la infraestructura desenvolupada i que demostri el correcte funcionament del projecte.

Construcció d'un prototipus Cal muntar el suport físic que implementarà tot el sistema (tàndem projector - càmera, gestionat amb un ordinador portàtil).

Cadascuna d'elles són petites metes que s'han d'assolir per tal d'arribar amb èxit a l'objectiu final. A més a més, s'ha de construir un prototipus que compleixi amb totes aquestes característiques teòriques[3].

1.4 Organització de la memòria

El projecte de final de carrera consta de dues parts ben diferenciades, la conjunció de les quals permet dur-lo a terme:

[2] Aquesta configuració automàtica inclou els processos de calibratge geomètric i cromàtic que es descriuen al capítol 7

[3] De fet, es pot considerar que la seva construcció és imperativa, atès que es vol comprovar que, efectivament, tot funciona correctament

Calibratge del sistema La infraestructura a la que el projecte pretén arribar requereix d'un calibratge acurat dels diferents components[4].

Desenvolupament d'una aplicació 3D El sistema de realitat virtual requereix una aplicació que aprofiti la infraestructura sobre la qual està corrent i que aprofiti al màxim tots els recursos disponibles.

Per aquest motiu, s'estima oportú que la memòria respecti aquesta diferenciació dels paquets de treball i que, per tant, s'organitzin en diferents capítols, on cadascun d'ells comprengui una de les fases.

1.4.1 El calibratge i el desenvolupament d'una aplicació

El fet de partir de dues fases que estan tan diferenciades facilita pensar que qualsevol d'elles és prou important per si sola i té suficient entitat com per a considerar-la un projecte auto contingut, de tal manera que seria la unió d'ambdós projectes la que representaria la totalitat del PFC. En aquest cas s'ha cregut convenient interpretar-ho d'aquesta manera, i s'ha decidit, doncs, donar-los-hi la categoria de projecte.

Aquesta decisió permet treballar-hi de forma independent, documentant-los per separat i fent que tasques com l'anàlisi de requisits, el disseny del sistema o la fase de proves es vinculin a cadascun d'ells sense que s'interfereixin. Això facilita enormement el desenvolupament del projecte, ja que la feina queda dividida des de l'inici en entitats més petites. A més a més, també en simplifica la seva comprensió, atès que s'estudia, en cada cas, una part més petita del total.

És important subratllar, però, que no estan completament desvinculats l'un de l'altre; de fet, i com ja s'ha insinuat anteriorment, el *desenvolupament de l'aplicació 3D* depèn del *calibratge del sistema*. Així, es pot intuir fàcilment que s'haurà de treballar pensant en què cap de les dues parts està aïllada totalment i que, conseqüentment, s'ha de definir molt bé el nexe d'unió entre elles.

1.4.2 Mètode de desenvolupament

Els nostres objectius impliquen treballar amb moltes parts totalment diferents de la informàtica. Com veurem més endavant, treballem amb xarxes, amb visió per computador, amb llenguatges diferents, etc. i moltes d'aquestes parts ens eren desconegudes abans de començar. Per tant, molta de la feina ha estat buscar eines per facilitar-nos al màxim el desenvolupament. Hem trobat moltes llibreries realment útils i d'altres que ens han servit d'alguna manera, però trobar-les i provar-les ha consumit bona part del nostre temps. També hem tingut feina per entendre nous conceptes i nous llenguatges, que mai havíem fet servir.

El mètode de desenvolupament ha estat durant tot el projecte: entendre el problema, cercar llibreries o eines per solucionar-ho, provar-les i intentar resoldre'l. Val a dir que molts cops esperàvem trobar eines molt fàcilment perquè pensàvem que el problema que volíem resoldre havia de ser un problema conegut i ,al final, després de molta cerca, la millor solució era desenvolupar la solució nosaltres mateixos.

1.4.3 Treball en equip

El treball en equip és un repte més en la consecució de qualsevol projecte, però aquest és més fàcilment tractable, ja que, com ja hem dit, consta de dos parts molt ben diferenciades, i

[4]Calibratge que inclou tant la correcció geomètrica com la cromàtica de les imatges generades pels mòduls projectors.

cada part té seccions també prou ben separades. Una de les tasques més importants abans de començar va ser distribuir bé què havia de fer cadascú per no interferir-nos i per maximitzar com aprofitem el temps. Podem dir finalment que estem molt satisfets del treball realitzat i de com hem treballat cadascú.

Capítol 2

Sistemes de realitat virtual

A continuació, es descriu què és un sistema de realitat virtual i quines tècniques hi ha per tal d'augmentar la sensació d'estar dins del mon virtual. També descriurem alguns sistemes actuals de realitat virtual, com la cave o la power wall. Abans, però començarem per definir uns quants conceptes per tal de poder comprendre-ho tot millor.

2.1 Què és un sistema de realitat virtual?

Es pot definir la realitat virtual com una tecnologia que permet a un usuari interaccionar amb un entorn simulat per ordinador, essent aquest real o imaginari. La majoria d'aquests entorns se centren en l'aspecte visual d'aquesta realitat, ja sigui a través de la pantalla d'un ordinador o mitjançant pantalles estereoscòpiques, tot i que cada cop més s'integra també la part auditiva i inclús d'altres sentits.

Un sistema de realitat virtual és, doncs, una solució construïda físicament i amb un programari adaptat a aquella construcció, que respecta la definició anterior.

2.2 Què entenem per sistema immersiu?

Un sistema és immersiu quan pretén aconseguir que l'usuari tingui la sensació d'estar dins d'aquest món simulat. Quan més aconsegueix aquesta sensació, més immersiu es diu que és el sistema. Ens adonem ràpid de que el que cal és que el sistema estimuli els sentits de l'usuari. Actualment, el sentit més habitual per estimular és la vista, però hi ha investigacions per cadascun dels cinc. En veurem uns exemples més endavant

2.2.1 Tècniques per aconseguir la immersió

Algunes tècniques visuals emprades per a aconseguir aquest realisme visual són la projecció d'imatges en estèreo, que vol dir que projectem una imatge per l'ull dret i una per l'esquerra, la qual cosa permet que l'usuari tingui sensació de profunditat. Aquesta projecció en estèreo s'aconsegueix de dues maneres:

Estèreo passiu La projecció en passiu consisteix en tenir dos projectors, associats a cada ull, projectant sobre la mateixa superfície. A cadascun se li posa entre la pantalla i la lent un filtre de polarització de llum, de manera que la imatge que arriba a la pantalla arriba i rebota polaritzada segons el filtre. Si s'utilitzen filtres diferents per cada projector,

quan rebotin a la pantalla nosaltres podrem utilitzar unes ulleres amb els mateixos filtres que els projectors i així cada ull veurà només la imatge que li correspon. Aquest tipus d'estereovisió és més assequible, però té alguns problemes. El més gran és que de vegades les ulleres no filtren prou bé i llavors es perd immersió. D'això en diem superposició d'imatges.

Estèreo actiu La projecció en actiu és una filosofia diferent. Consisteix en projectar alternativament una imatge per l'ull dret seguit d'una de l'ull esquerra. La manera que té l'usuari de distingir i separar-les ve a través de les ulleres, que estan constantment tapant l'ull que toca. Per decidir quin ull tapar, les ulleres es sincronitzen amb els projectors. D'aquí ja podem veure dos coses: el projector haurà de ser bo perquè haurà de suportar aquesta velocitat de canvi, i les ulleres seran sofisticades, i per tant cares, perquè han de ser capaces de sincronitzar-se i tapar i destapar un ull a una velocitat prou gran. Com a avantatges té que és immediat de calibrar i no té problemes de superposició d'imatges

És important remarcar que l'estereovisió ha de tenir en compte la posició de l'espectador respecte la pantalla, perquè la distància i la posició de la càmera en el mon virtual ha de correspondre aproximadament amb la realitat. Si no es fes, l'usuari podria tenir una sensació de mareig, degut a que el que està veient no es correspon amb el que hauria de veure, o pèrdua d'immersió ja que veuria la imatge deformada o unes distàncies molt inadequades. Una aplicació pot suposar l'espectador estàtic respecte a la pantalla o fer un seguiment del seu cap. Del seguiment de la posició i orientació del cap de l'usuari se'n diu *head tracking*.

Paradoxalment, centrant-nos en els sistemes de realitat virtual visuals s'observa que no es necessita que tinguin un nivell de detall molt elevat (nombre de polígons, textures, il·luminació, etc.) per tal d'immergir l'usuari són més importants factors com el *head tracking*, les ombres, reflexions, etc.

Tenint això en compte, ja fa temps que s'ha començat a investigar en altres mètodes per millorar l'immersió utilitzant d'altres sentits. Un dels sentits que més s'empra és l'oïda. Avui en dia els sistemes de so envoltant són prou comuns i estan suficientment desenvolupats industrialment que són assequibles per tothom.

També s'està investigant amb el sentit del tacte, amb els dispositius que en diem hàptics. Una de les aproximacions més comuns és un braç robòtic amb un extrem que s'agafa amb la mà. L'usuari pot agafar aquest extrem per moure el braç robòtic, i un programari ensenyarà visualment la posició on es troba la mà dins del mon virtual, i si aquesta impacta amb algun objecte virtual, el braç robòtic oferirà resistència en aquella direcció, donant una sensació de tacte. D'aquesta resistència que fa el braç segons li digui el programari en diem *force feedback*.

Aquests sistemes encara tenen bastants problemes i inconvenients. Per començar, el braç robòtic és massa car, i per tant només assequible als centres d'investigació. Després que per notar i poder resseguir una superfície virtual, la detecció de col·lisions ha de ser molt ràpida, sinó l'usuari notarà un tremolor que farà desaparèixer la sensació d'immersió. Això és degut a que el tacte és molt més sensible que la vista en el que es refereix a canvis. En veurem un exemple més endavant. Tot i tenir aquests problemes, tenen molt potencial, sobretot en aplicacions com la medicina, el disseny de cotxes, treball a distància, etc. i per aquesta raó es continua desenvolupant intensament aquesta via de sistemes immersius.

Per últim, també s'ha investigat amb el sentit de l'olfacte, però és una via molt poc avançada. S'han construït alguns prototipus, i proposat idees com integrar sensors i emissors d'olfacte en mòbils, dispositius connectats a l'ordinador que emeten olors, fins i tot webs que suportin aquests dispositius i enviïn informació per utilitzar-los. Les dificultats tècniques d'aquests aparells, i també la del programari per utilitzar-los fan que encara sigui una idea del futur. També hi ha un gran potencial al darrera, perquè el sentit de l'olfacte és un gran estimulant al

cervell, és el sentit que més records i reaccions tant agradables com desagradables evoca a les persones.

2.3 Exemples

Tot seguit veurem uns quants exemples dels sistemes de realitat virtual que hem anat comentant al llarg del capítol. Començarem descrivint el sistema en el qual es basa el nostre projecte, la *cave*.

2.3.1 La *cave*

Un sistema cave està format típicament per tres pantalles i el terra formant una habitació molt semblant a la de la figura 2.1. En aquestes pantalles una aplicació hi projectarà un mon virtual, i l'usuari que estigui dins de l'habitació, al estar envoltat d'aquestes imatges, tindrà la sensació d'estar dins d'aquest mon.

Figura 2.1: Cave.

Per millorar la sensació d'immersió (la sensació d'estar dins del mon virtual), les caves van evolucionar i van incorporar estereovisió, amb *head tracking*. Actualment també solen incorporar sistemes de so envoltant.

2.3.2 La *Power Wall*

La *power wall* es podria descriure com l'evolució d'un cinema tradicional. Es tracta d'una pantalla tan gran com es pugui i els espectadors estan asseguts a cadires. S'utilitza normalment estereovisió passiva i també so envoltant. Les aplicacions que l'utilitzin poden suposar que l'usuari ho esta veient en el centre del grup de cadires, i tot i que a la realitat això no sigui exacte, és una aproximació vàlida perquè típicament són espectadors estàtics.

2.3.3 Ulleres de realitat virtual

Les ulleres de realitat virtual van ser una de les primeres aproximacions de sistemes immersius que es van construir. La idea es tenir dues pantalles molt petites separades i que cada ull nomes en pugui veure una. Aquestes pantalles es munten sobre unes ulleres i el *headtracking* es pot fer mesurant els girs del cap amb acceleròmetres i es suposa que l'usuari està quiet a una cadira, o es pot mesurar també la posició amb un sistema com el de la cave i llavors l'usuari es pot moure. Els inconvenients són que les mini pantalles han de ser adequades perquè han de

Figura 2.2: Power Wall.

tenir una resolució prou alta i ser suficientment grans, o sinó l'usuari haurà de forçar la vista i la qualitat de la imatge no serà bona. També hi ha d'altres problemes molt més difícils de solucionar, com que l'aplicació típicament només té coneixement de la posició del cap, però no de la resta del cos, per tant, no el pot ensenyar, fet que provoca una sensació de mareig que pot ser molt desagradable. A l'actualitat, s'està investigant l'evolució d'aquestes ulleres que eliminaran aquests problemes. La idea és utilitzar unes ulleres normals, però incloure uns petits projectors a les patilles. D'aquesta manera, i fent *headtracking*, podem solucionar el problema de que l'usuari es maregi per no veure el seu propi cos. També es solucionarien els problemes de que la pantalla no és prou gran. A 2.3 la figura podem veure unes ulleres de realitat virtual actuals.

Figura 2.3: Ulleres de realitat virtual.

2.3.4 Dispositius hàptics

Com ja hem dit, els dispositius hàptics es centren en el sentit del tacte de l'usuari. El típic exemple de d'aquests és un braç robòtic hàptic com el de la figura 2.3.4. L'usuari agafa la punta més fina amb la mà i la va movent lliurement fins que troba un objecte i nota l'impacte.

Figura 2.4: Braç robòtic hàptic.

També podem trobar d'altres tipus, com mans robòtiques connectades a distància amb una altra mà robòtica que té l'usuari. Aquest podrà moure la seva mà, que està lligada amb la robòtica i notarà el que estigui tocant la mà robòtica a distància. La figura següent és auto-explicativa.

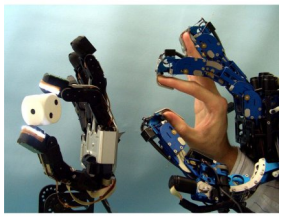


Figura 2.5: Robot hàptic petit.

Per últim, s'està començant a fabricar braços robòtics hàptics més petits per ser més assequibles i més pràctics. Òbviament les prestacions no seran les mateixes, però si troben mercat, la progressió pot ser molt ràpida. La figura següent n'és un exemple.

Figura 2.6: Braç robòtic hàptic assequible.

Capítol 3

Propòsit del projecte

Al primer capítol d'aquesta memòria hem descrit a grans trets el projecte i hem esbossat els objectius que persegueix, sempre de forma introductòria i sense entrar en detalls.

Aquí, en canvi, es veurà amb més definició l'entorn en què ens movem i comprendrem millor el seu abast. Això implicarà analitzar quin és el context actual en el món dels sistemes de realitat virtual, com el nostre projecte pretén canviar-lo o discutir les alternatives existents que podem trobar-nos.

Finalment, també es parlarà de les persones que es veuen afectades pel projecte, descrivint el seu rol i el nivell d'implicació que hi tindran.

3.1 Context actual

Al capítol 2 es parla de què és un sistema de realitat virtual i algunes de les tècniques emprades actualment per tal de millorar la immersió que ofereix. També es poden veure alguns exemples de sistemes de realitat virtual, tals com una *cave* o una *power wall*.

Ara bé, on se situa el nostre projecte? Quina aplicabilitat té? Què pretén millorar? Aquestes són algunes de les preguntes que aclarirem en aquesta secció i les successives.

Els sistemes de realitat virtual que hi ha avui en dia no estan a l'abast del públic general, atès que acostumen a tenir uns costos molt elevats i requerir d'unes instal·lacions específiques. El nostre projecte, per tant, aconseguiria apropar aquest tipus de sistemes a un ventall de gent molt més ampli.

A més a més, tenint en compte que un dels objectius a acomplir és la *portabilitat* del sistema, permetria alliberar-nos de la dependència de disposar d'un emplaçament fix determinat, essent una solució interessant, també, per a aquelles entitats que actualment ja disposen de sistemes de realitat virtual "clàssics".

De fet, la idea d'aquest PFC neix amb l'objectiu de dissenyar un sistema que pugui competir amb una *cave*, solucionant-ne els problemes que veurem tot seguit.

3.2 Problema a resoldre

Una *cave* consisteix en «una sala» de $3 \times 3 \times 3m$ on es projecten imatges a quatre cares del cub (tres parets i un terra). L'usuari se situa a l'interior d'aquest espai i s'endinsa, d'aquesta manera, al món virtual.

Tenint en compte que l'ideal que persegueix el nostre projecte és esdevenir el substitut d'una *cave*, és necessari comprendre quins són els principals problemes que aquesta té:

- La ≪sala≫ (la *cave* en si) ha d'estar ubicada a un emplaçament molt més gran que permeti ubicar-hi les pantalles de projecció, els projectors i l'equip informàtic.
- Per tal d'evitar les ombres que produiria l'usuari al passar entre el projector i la pantalla de projecció, la imatge és retroprojectada. Això vol dir que les pantalles han de ser especials per a què la imatge es pugui veure amb prou lluminositat.
- Els projectors han de ser capaços d'enviar imatges amb una resolució força gran, i projectar en superfícies de $3 \times 3m$. Es tracta, doncs, de projectors d'una gran envergadura i d'elevat cost.
- Atès el cost dels projectors, no és viable plantejar-se tenir-ne dos per cada pantalla projectada (essent, d'aquesta manera, un total de vuit). Per tant, per tal de generar imatges estèreo s'ha d'utilitzar l'*estèreo actiu*; això implica que cada ull només veu les imatges la meitat del temps, amb la conseqüent pèrdua de lluminositat que això implica.

Com es pot veure, l'àmbit d'ús d'una *cave* és molt reduït, tractant-se d'una solució molt restrictiva: la portabilitat d'aquest sistema (entenent portabilitat com la possibilitat de desmuntar-lo i muntar-lo en una altra ubicació) és pràcticament nul·la.

De fet, aconseguir un substitut de la *cave* és una tasca titànica, i més tenint en compte els objectius que volem acomplir (vegeu la pàgina 2). Ara bé, es pot considerar que el PFC en si es perfila com el predecessor d'un sistema molt més ampli i ambiciós, el qual sí que pot arribar a competir, realment, amb una *cave*.

Amb els termes en què es planteja ara mateix el projecte, però, no es disposa d'una pantalla tant gran com pot ser la d'una *cave*, ni tampoc de més d'una paret que envolti a l'usuari. Tampoc s'hi implementa la visió en estèreo.

De totes maneres, aconseguir projectar en una paret tant gran no suposa un repte excessivament gran (vegeu []), ni tampoc la visió estèreo[1]. La tasca més costosa seria la de projectar en més d'una paret (les quals, recordem, estan disposades de forma arbitrària), però això cau fora de l'abast d'aquest projecte.

3.3 Resultats esperats

Ja hem vist que ara mateix no pretenem dissenyar un sistema que sigui el substitut d'una *cave*, però sí que en seria el predecessor.

El que volem és un sistema que sigui *altament portable*, que funcioni en *qualsevol lloc* de *forma automàtica* i que tingui un bon nivell d'*immersió*. Tot plegat ho aconseguirem assolint els objectius descrits a l'apartat 1.3.

En qualsevol cas, acomplir aquests objectius vol dir solucionar els principals problemes que planteja una *cave*, o molts dels sistemes de realitat virtuals d'avui en dia: la portabilitat i el cost de la infraestructura.

Al capítol 7 descrivim les tècniques utilitzades per tal d'aconseguir que la infraestructura desenvolupada es calibri a una ubicació qualsevol donada de forma automàtica, de tal manera que la intervenció de l'usuari sigui pràcticament nul·la.

Al capítol 9, en canvi, demostrarem el funcionament d'aquesta infraestructura que es calibra per si sola muntant-hi una aplicació a sobre que la utilitzi, a part d'integrar-hi elements com el *head tracking*.

[1] Atès que el nostre projecte admet la projecció d'imatges sobre qualsevol paret, s'hauria d'emprar l'estèreo actiu (el passiu és inviable, ja que perdríem la polarització de la llum). La utilització de l'estèreo actiu no representa cap dificultat teòrica, i només requereix la generació d'imatges diferents per a cada ull; no s'hauria de calibrar el projector de l'ull esquerre amb el del dret.

Amb tot això, assentarem les bases del que, més endavant, pot evolucionar cap a un sistema molt més gran i ambiciós.

3.4 Alternatives al projecte

Actualment, no hi ha sistemes de realitat virtual que disposin de les característiques amb que volem dotar el nostre projecte. El principal handicap que tenen aquests sistemes és la portabilitat que, o és pràcticament inexistent, o no són tant flexibles com el que presentem.

3.5 *Stakeholders* i usuaris

Els *stakeholders* són tota entitat afectada o necessària pel desenvolupament i ús d'un sistema. Tractant-se d'un projecte de realitat virtual i caire certament acadèmic, el nombre d'*stakeholders* que hi ha és força reduït.

Els rols que s'han identificat són:

Administrador És aquella persona que s'encarrega de *muntar* i *configurar* la infraestructura del sistema.

Controlador És aquella persona que supervisa l'ús que l'usuari està fent del sistema.

Usuari És la persona que utilitza el sistema i que, per tant, s'"endinsa" en el món virtual.

Es pot considerar que l'*administrador*, tal qual s'ha definit, està, de fet, realitzant dos rols diferents: el que s'encarrega de la part física de la infraestructura, muntant els projectors, les càmeres, etc., i el que s'encarrega de la part programàtica del sistema, configurant-lo convenientment (executant els processos de calibratge, l'aplicació d'usuari, etc.). Per altra banda, el *controlador* és una figura que pot ser-hi present o no, ja que la seva participació no és estrictament necessària.

Capítol 4

Planificació del projecte

La *planificació* d'un projecte és una de les parts més importants de la *gestió de projectes*.

El primer que cal fer és definir l'àmbit del projecte[1], de tal manera que, a continuació, es puguin definir les tasques que s'hauran d'assolir per a completar-lo, així com la relació que hi ha entre elles. Tot plegat fa possible estimar la duració total que necessitarà el projecte, determinar les tasques crítiques que si duren més d'allò que s'ha previst provocarien que s'acabés més tard, calcular el cost que suposarà desenvolupar-lo o coordinar els recursos que es necessitaran.

En aquest capítol es mostren les diferents tasques en què hem decidit descompondre el projecte i les seves dependències, qui s'encarrega d'elles i les fites que s'han d'assolir.

4.1 Introducció

El projecte s'inicia formalment a mitjans de juliol de 2008, moment en què es matricula, tot i que el procés d'estudi i desenvolupament va iniciar-se abans, a principis de l'any 2008, que és quan es contacta amb Pere Brunet i es decideix inscriure'l a la facultat.

Aquest capítol inclou totes les tasques realitzades des del moment en què es va inscriure fins a la seva fi.

4.2 Tasques a dur a terme

La taula que tenim a continuació mostra totes les tasques en què s'ha dividit el projecte i indica, per cadascuna d'elles, qui és el responsable de fer-la. Les tasques es corresponen a les del diagrama de Gantt de la figura 4.2.

Nom	**Assignat a**
Tasques administratives	
Matriculació del PFC	Ambdós
Redacció informe previ	Ambdós

(continua a la següent pàgina)

[1] Vegeu el capítol 3

(ve de la pàgina anterior)

Entrega informe previ	Ambdós
Redacció memòria	Ambdós
Revisió memòria	Ambdós
Correccions memòria	Ambdós
Impressió i enquadernació memòria	Ambdós
Verificació i entrega e la memòria	Ambdós
Pactar data de presentació	Ambdós
Preparació de la presentació	Ambdós
Presentació	Ambdós
Estudi previ	
Lectura dels *papers*	Ambdós
Definició d'objectius del projecte	Ambdós
Especificació del mòdul de calibratge	
Acotar objectius del mòdul	Ambdós
Definició dels casos d'ús	David
Anàlisi de requisits	David
Definició de les classes	David
Disseny del mòdul de calibratge	
Estudi d'integració xarxa al codi Java	David
Modificació diagrama classes per integrar-hi patrons	David
Definició dels algorismes de coordinació	David
Desenvolupament calibratge geomètric	
Implementació	
Impl. de les classes	Ambdós
Impl. dels algorismes de visió per computador	Rodrigo
Impl. del mòdul de xarxa	David
Impl. dels algorismes de calibratge	Ambdós
Integració dels components	Ambdós
Testos	
Test conjunt dels components	David
Proves de calibratge geomètric	David
Desenvolupament calibratge cromàtic	
Implementació	
Impl. de les classes	David
Impl. dels algorismes de calibratge	David
Integració amb el mòdul geomètric	David
Testos	
Test de l'algorisme de calibratge	David
Test amb calibratges reals	David
Correccions necessàries al calibratge	
Correcció del calibratge geomètric	David
Correcció del calibratge cromàtic: intensitats	David
Correcció del calibratge cromàtic: zona d'intersecció	David
Escriptura de resultats	
Definició de resultats a compartir	Ambdós
Proves d'escriptura	Ambdós
Especificació de l'aplicació d'usuari	

(continua a la següent pàgina)

(ve de la pàgina anterior)

Acotar objectius del mòdul	Ambdós
Definició dels casos d'ús	David
Anàlisi de requisits	David
Estudi de l'aplicació [CL]	Rodrigo
Definició de les classes	Rodrigo
Disseny de l'aplicació d'usuari	
Estudi dels diversos patrons de xarxa	Rodrigo
Modificació diagrama classes per integrar-hi patrons	Rodrigo
Desenvolupament del mòdul de visualització	
Implementació	
Estudi de C#	Rodrigo
Reorganització del codi	Rodrigo
Integració de la xarxa	Rodrigo
Lectura dels resultats del calibratge	Rodrigo
Implementació dels *shaders*	Rodrigo
Proves	
Proves de xarxa	Rodrigo
Proves de *shaders*	Rodrigo
Proves de tota l'aplicació	Rodrigo
Prototipus	
Estudi de les necessitats necessitats	Ambdós
Adquisició / reutilització de materials	Ambdós
Construcció dels prototipus	Ambdós

Taula 4.1: Tasques a realitzar des de la matriculació del projecte fins al final.

4.3 Planificació estimada

La primera planificació que es va dur a terme no plantejava les tasques en què s'hauria de dividir l'*aplicació d'usuari*, i per aquest motiu el diagrama de Gantt de la figura 4.1 estima tot el procés com una única tasca la duració de la qual és d'uns, aproximadament, dos mesos.

4.4 Planificació corregida

A mitjans de juliol de 2008, moment en què es matricula el projecte, es decideix refer el diagrama de Gantt, per tal de, per una banda, incloure-hi els retards que hi havien hagut durant els primers mesos de desenvolupament i, per l'altra, de detallar les tasques que composen la segona part del projecte. El diagrama de la figura 4.2 mostra les següents particularitats:

- S'han adaptat les estimacions de les tasques dutes a terme a les seves duracions reals.
- S'adapten els inicis de la resta de tasques, degut a què les duracions inicialment previstes de les primeres tasques s'han estès.
- S'hi introdueix la planificació de la segona part del projecte: l'aplicació d'usuari.

4.5 Duració real

Finalment, la figura 4.3 mostra un diagrama amb la duració real que ha tingut el projecte, per tal de poder-ho comparar amb la planificació que s'havia fet.

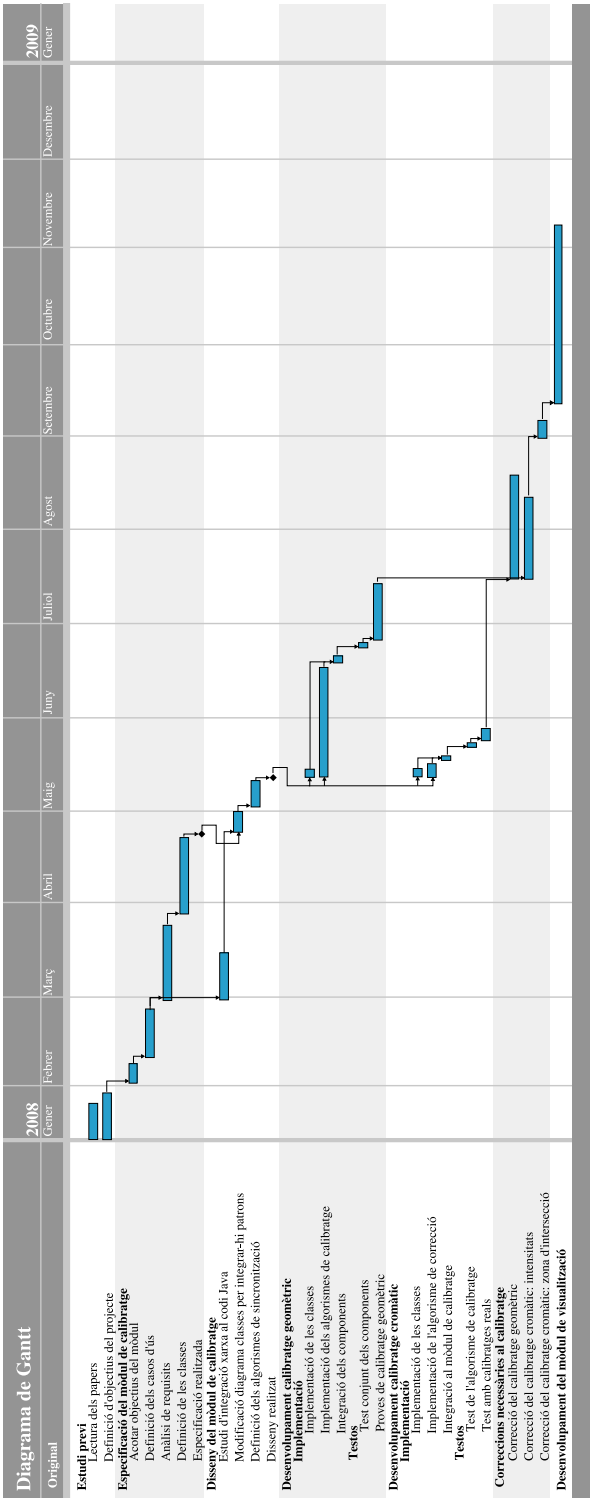


Figura 4.1: Diagrama de Gantt amb la planificació estimada a l'inici del projecte, quan es va inscriure.

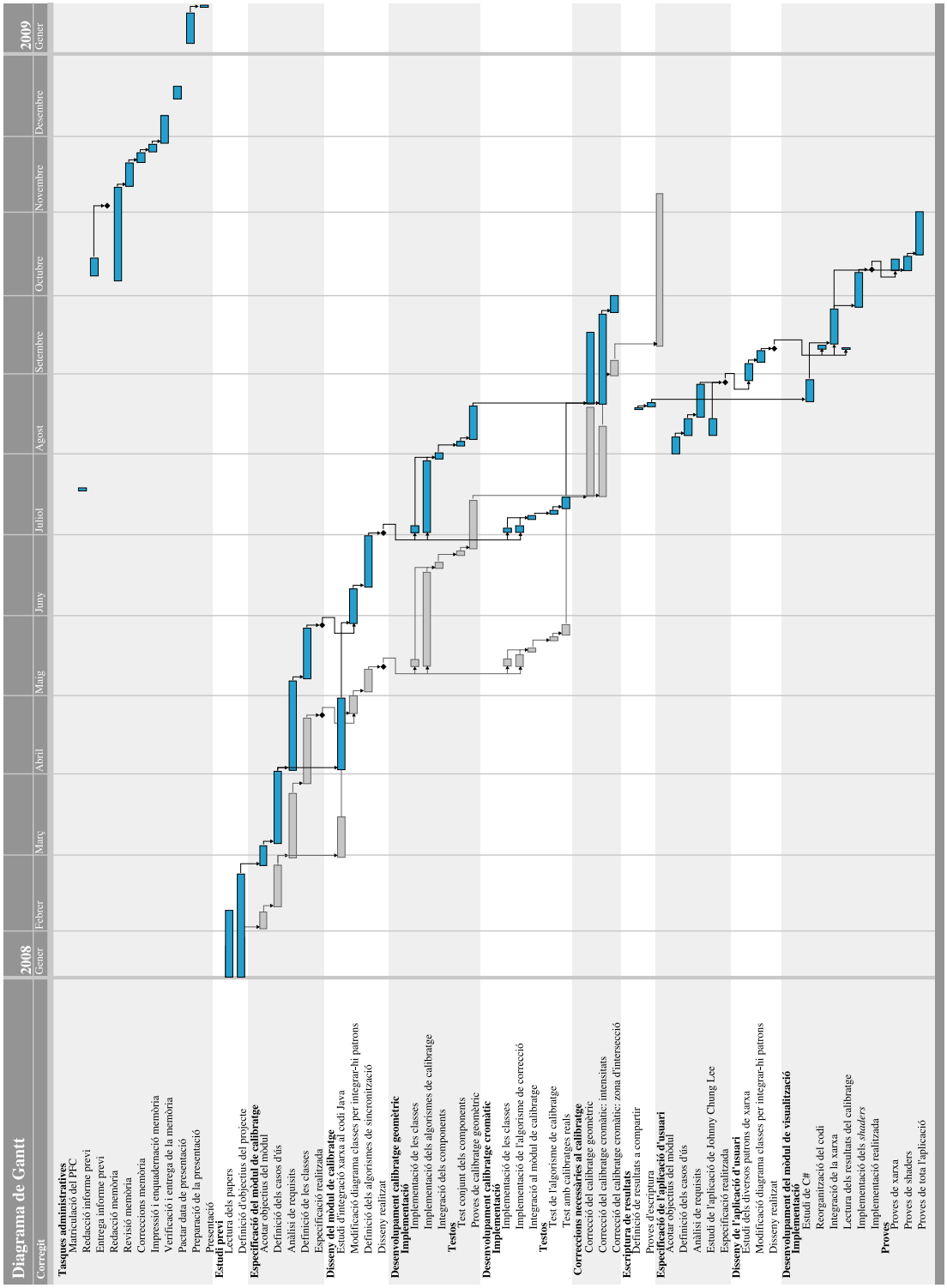


Figura 4.2: Diagrama de Gantt actualitzat al moment de matricular el projecte, corregint les duracions.

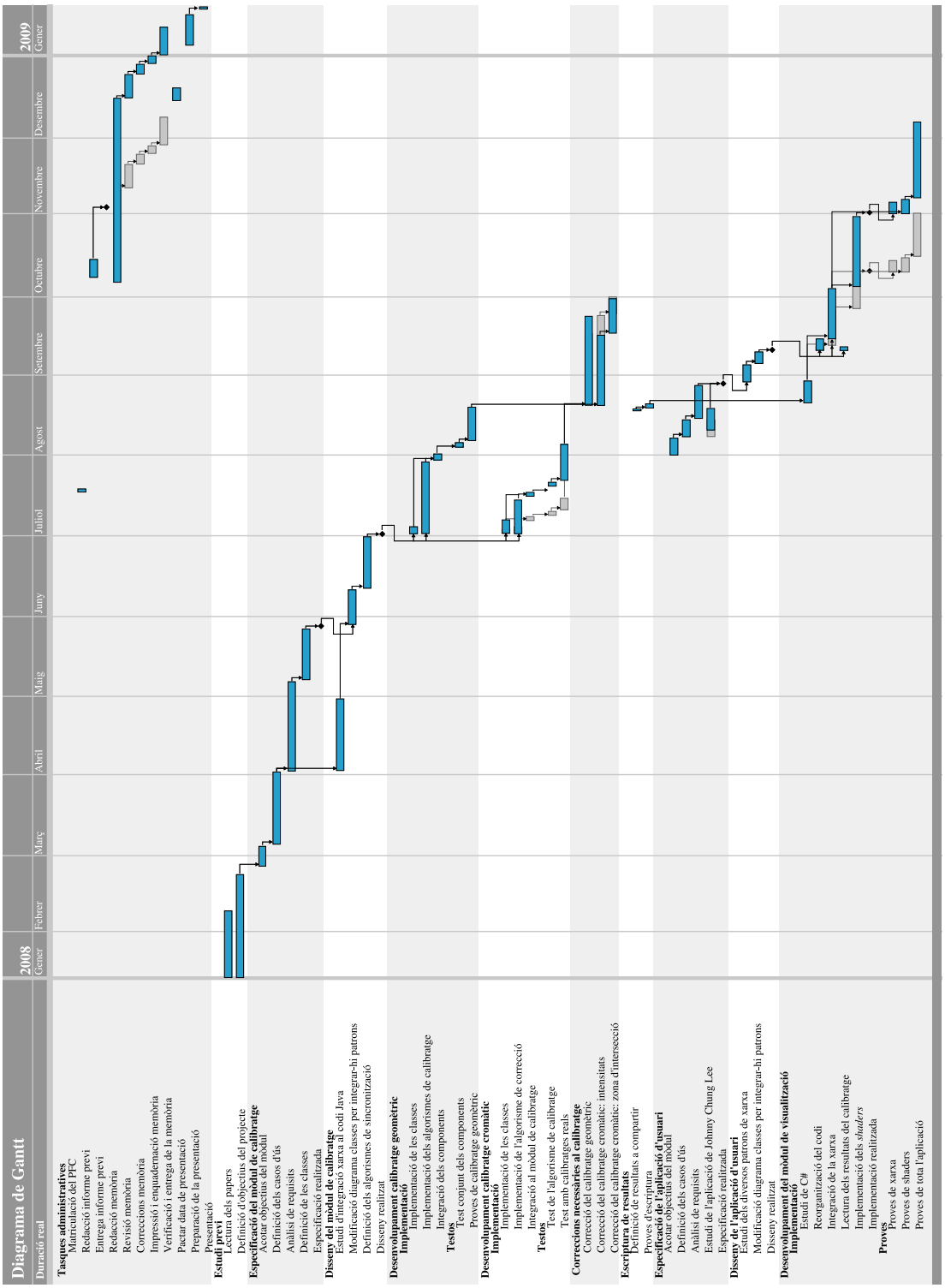


Figura 4.3: Diagrama de Gantt final, on es mostra la durada real de cadascuna de les tasques.

Capítol 5

Homografies

En aquest apèndix s'explica què són les homografies, per a què serveixen i com encaixen dins d'aquest projecte.

També s'explica amb força detall com s'ha implementat el seu càlcul amb les eines disponibles al sistema (càmeres i projectors), comentant les diferents iteracions que s'han dut a terme per a arribar a la solució final. Això vol dir que es comença descrivint com calcular una homografia quan només es té un projector i una càmera, i com s'acaba arribant a fer-ne el còmput amb N projectors i N càmeres.

5.1 Què són les homografies?

Una *homografia* és un concepte emprat en l'àmbit de la geometria projectiva. Aquesta defineix la relació que hi ha entre dues imatges, de tal manera que donat un punt qualsevol d'una d'elles, aquest correspon a un, i només un, punt de l'altra.

Aquesta correspondència entre dos punts se sol expressar amb una matriu H que té la següent forma:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Sigui $P = (x_p, y_p, 1)$ un punt qualsevol que pertany a la primera figura i $Q = (x_q/r, y_q/r, 1)$ un altre que pertany a la segona, i saben que entre aquests dos punts existeix la correspondència abans esmentada, es té que:

$$\begin{pmatrix} x_q \\ y_q \\ r \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix}$$

5.2 Com calcular una homografia

Per tal de calcular una homografia pla-a-pla es poden emprar dos mètodes, la *solució lineal no homogènia* i la *solució homogènia*. A continuació es descriuen ambdós mètodes.

5.2.1 Mètode d'estimació homogènia

Escrivint l'homografia H en forma de vector $h = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^T$, les equacions homogènies per a n punts esdevenen $Ah = 0$, essent A la matriu de mida $2n \times 9$ següent:

$$A = \begin{pmatrix} x_{p_1} & y_{p_1} & 1 & 0 & 0 & 0 & -x_{p_1}x_{q_1} & -y_{p_1}x_{q_1} & -x_{q_1} \\ 0 & 0 & 0 & x_{p_1} & y_{p_1} & 1 & -x_{p_1}y_{q_1} & -y_{p_1}y_{q_1} & -y_{q_1} \\ x_{p_2} & y_{p_2} & 1 & 0 & 0 & 0 & -x_{p_2}x_{q_2} & -y_{p_2}x_{q_2} & -x_{q_2} \\ 0 & 0 & 0 & x_{p_2} & y_{p_2} & 1 & -x_{p_2}y_{q_2} & -y_{p_2}y_{q_2} & -y_{q_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{p_n} & y_{p_n} & 1 & 0 & 0 & 0 & -x_{p_n}x_{q_n} & -y_{p_n}x_{q_n} & -x_{q_n} \\ 0 & 0 & 0 & x_{p_n} & y_{p_n} & 1 & -x_{p_n}y_{q_n} & -y_{p_n}y_{q_n} & -y_{q_n} \end{pmatrix}$$

El vector h que minimitza els residus de $\det(A)$, essent $\det(H) = 1$, és obtingut pel vector propi del menor valor propi d'$A^T A$. Aquest vector propi es pot obtenir directament de la descomposició de valor singular (SVD, per les seves sigles en anglès) d'$A^T A$.

5.2.2 Solució lineal no homogènia

En aquest mètode, un dels elements de l'homografia té un valor fixat, generalment la unitat, computant la solució dels altres vuit elements via una pseudo inversa. Com a desavantatge es té que s'obtenen resultats força pobres si el valor de l'element escollit hauria d'haver sigut zero.

En el cas que es discuteix en aquest projecte, quan diem que $Q = HP$, es considera que el símbol d'igualtat vol dir *igualtat ignorant el factor d'escalat*. Aquest no afecta a l'equació i, per tant, només vuit dels valors d'aquesta homografia són significatius.

Per tant, si assumim que

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$$

per tal de calcular els vuit elements que manquen d'H només caldrà resoldre el següent sistema:

$$\begin{pmatrix} x_{p_1} & y_{p_1} & 1 & 0 & 0 & 0 & -x_{p_1}x_{q_1} & -y_{p_1}x_{q_1} \\ 0 & 0 & 0 & x_{p_1} & y_{p_1} & 1 & -x_{p_1}y_{q_1} & -y_{p_1}y_{q_1} \\ x_{p_2} & y_{p_2} & 1 & 0 & 0 & 0 & -x_{p_2}x_{q_2} & -y_{p_2}x_{q_2} \\ 0 & 0 & 0 & x_{p_2} & y_{p_2} & 1 & -x_{p_2}y_{q_2} & -y_{p_2}y_{q_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{p_n} & y_{p_n} & 1 & 0 & 0 & 0 & -x_{p_n}x_{q_n} & -y_{p_n}x_{q_n} \\ 0 & 0 & 0 & x_{p_n} & y_{p_n} & 1 & -x_{p_n}y_{q_n} & -y_{p_n}y_{q_n} \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x_{q_1} \\ y_{q_1} \\ x_{q_2} \\ y_{q_2} \\ \vdots \\ x_{q_n} \\ y_{q_n} \end{pmatrix}$$

En cas de què el sistema estigui sobre determinat (és a dir, $n > 4$), s'ha de calcular la solució utilitzant mínims quadrats.

5.3 Càlcul amb un projector i una càmera

Aquest és el cas més senzill amb el que podem treballar. L'objectiu serà associar les coordenades dels punts que es veuen a la càmera amb les coordenades que s'han enviat.

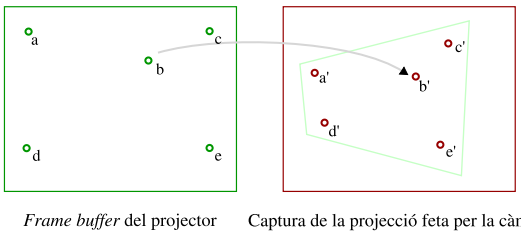


Figura 5.1: Correspondència entre punts enviats per un projector i punts capturats per una càmera.

Quan es disposa de la informació de correspondències s'aplica el mètode vist anteriorment per tal de computar l'homografia H. Amb ella, donat un punt qualsevol P_p del projector, podem saber el punt P_c associat que obtindríem en el sistema de coordenades de la càmera.

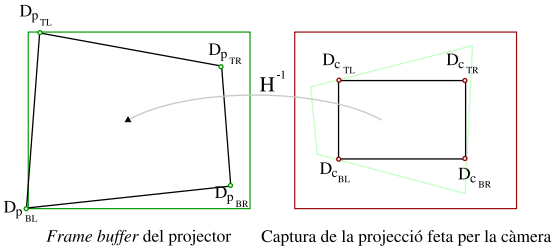


Figura 5.2: Càlcul de les coordenades de la pantalla final per a un projector i una càmera.

Tenint en compte que l'objectiu és construir una pantalla quadrada des del punt de vista de la càmera, es poden calcular les coordenades dels quatre vèrtexs (D_{cTL}, D_{cTR}, D_{cBL}, D_{cBR}) en el sistema de coordenades d'aquesta i aplicar H^{-1} per a conèixer quina zona del *frame buffer* del projector s'hauria de pintar (vegeu la figura 5.2).

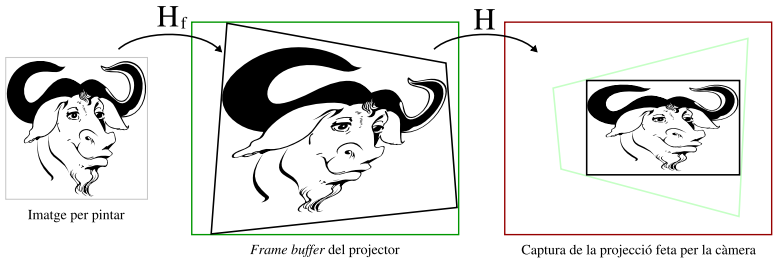


Figura 5.3: Diagrama de les transformacions que apliquem a la imatge inicial per a què al ser capturada per la càmera es vegi correctament, sense deformacions.

Finalment, es calcula una nova homografia H_f que converteixi les coordenades d'una imatge normalitzada ($x \in [0-1]$ i $y \in [0-1]$) cap a les coordenades (D_{pTL}, D_{pTR}, D_{pBL}, D_{pBR}). Tal i com es pot veure a la imatge 5.3, això permet que quan la imatge sigui projectada pel projector i capturada per la càmera, el que aquesta última vegi sigui la imatge inicial, sense deformacions de perspectiva[1].

[1] Això no vol dir que no hi pugui haver deformacions de relació d'aspecte.

5.4 Càlcul amb dos projectors i una càmera

A continuació, s'afegeix un nou projector al muntatge descrit anteriorment, complicant-lo lleugerament. Tal i com es pot apreciar a la figura 5.4, ara la càmera capta els patrons de dos projectors i, per tant, es disposa de la geometria de les dues àrees projectades (en verd i vermell).

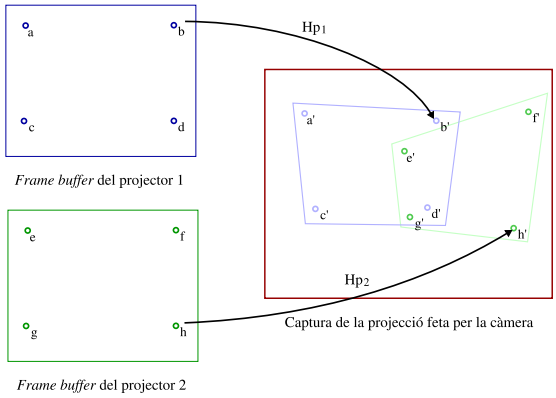


Figura 5.4: Correspondència entre punts enviats per dos projectors i punts capturats per una càmera.

De manera anàloga al cas anterior, es pot calcular una homografia H_{P_1} que converteixi els punts del *frame buffer* del projector P_1 a coordenades de la càmera, així com una homografia H_{P_2} que faci el mateix per al projector P_2. Tot seguit, caldrà calcular la geometria de la pantalla on es dibuixaran les imatges, tenint en compte que l'àrea que la contindrà és la unió de les àrees projectades per cada mòdul (vegeu la figura 5.5).

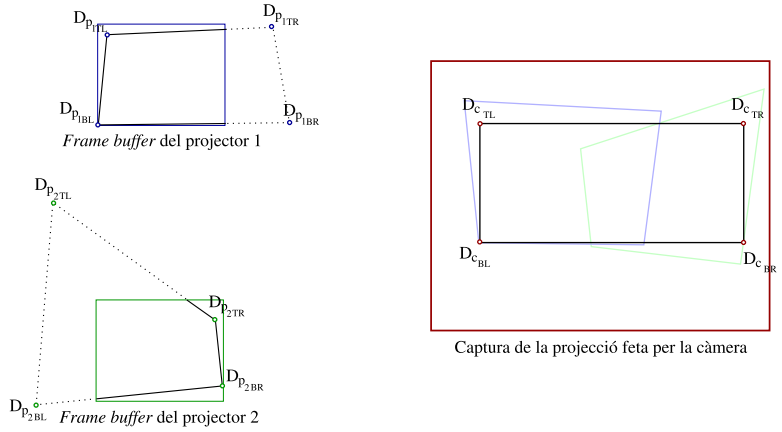
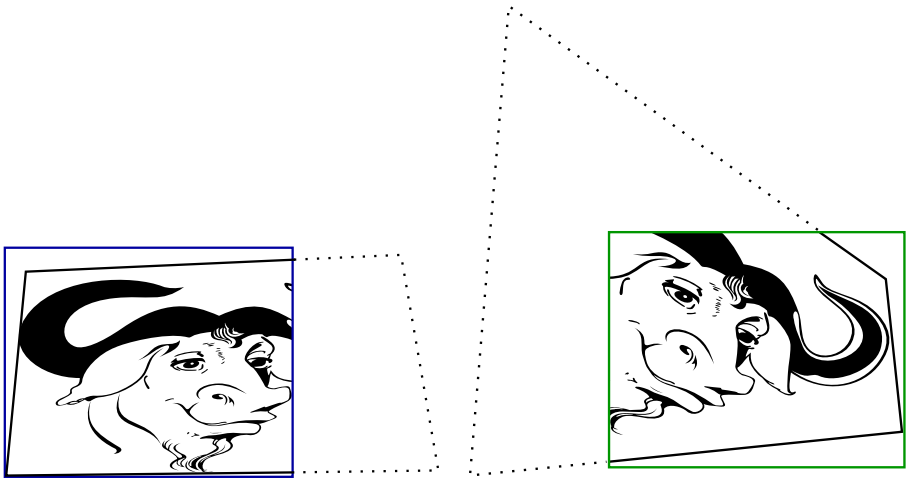


Figura 5.5: Càlcul de les coordenades de la pantalla final utilitzant dos projectors i una càmera.

Aplicant les homografies inverses es poden obtenir les coordenades (D_{cTL}, D_{cTR}, D_{cBL}, D_{cBR}) de la pantalla en els respectius sistemes de coordenades de cada projector. En aquest cas, és possible que algunes d'aquestes coordenades "quedin fora" del *frame buffer*. Això és totalment normal, atès que ara un projector només dibuixarà, en general, una part de la imatge, donant per fet que la resta de la imatge la pintarà l'altre.

Finalment, només cal calcular les homografies finals H_{fP_1} i H_{fP_2} que, de la mateixa manera

que en el cas d'un projector i una càmera, situaran la imatge correctament dins dels *frame buffers*.



(a) Imatge final situada a l'interior del *frame buffer* del projector 1.

(b) Imatge final situada a l'interior del *frame buffer* del projector 2.

Figura 5.6: Imatge final situada a l'interior dels *frame buffers*.

5.5 Càlcul amb N projectors i una càmera

Si se suposa que tenim N projectors enlloc de dos, el problema a resoldre és el mateix i no suposa cap complicació addicional.

En aquest cas, és necessari calcular les N homografies que ens permetran passar les coordenades de la pantalla final (calculades en el sistema de coordenades de la càmera) a les coordenades de cadascun dels projectors i computar, aleshores, les homografies finals que situïn una imatge normalitzada a cadascun dels N *frame buffers*.

Ara bé, cal tenir en compte que quan el nombre de projectors va augmentant, les dimensions que podrà assolir la pantalla final també creixen. Disposar d'una única càmera implica que arribarà un punt en què aquesta no tindrà un camp de visió prou ampli com per a veure tota l'àrea projectada. Tot i que allunyant-la s'aconseguiria veure-ho tot, la resolució de la càmera no permetria fer-ho amb prou detall.

La manera de solucionar-ho consisteix en disposar de més d'una càmera, de tal manera que cadascuna d'elles visualitzi una petita porció del total d'àrea projectada.

5.6 Càlcul amb N projectors i M càmeres

El cas més general consisteix en disposar d'N projectors i M càmeres, de tal manera que es podrà construir una pantalla tan gran com es desitgi.

Hem vist que si s'utilitza un elevat nombre de projectors i es pretén alinear-los utilitzant una única càmera, aquesta ha d'estar situada molt lluny per a què pugui capturar tota l'àrea projectada, perdent el nivell de detall que ens ofereix cada captura. Utilitzar més càmeres, en canvi, fent que cadascuna d'elles capturi només una part d'aquesta àrea, permet situar-les més a prop i, d'aquesta manera, aprofitar millor la resolució que tenen.

El càlcul de la geometria de la pantalla continguda dins d'un conjunt d'àrees projectades té una particularitat evident que ara cal subratllar: s'ha realitzat en un únic sistema de coordenades. Quan només es disposava d'una càmera, i aquesta realitzava les captures de cada projector, s'obtenia la geometria que projectava cadascun d'ells en un sistema de coordenades SC_{cam}. El fet de tenir totes les àrees representades al mateix espai permetia computar fàcilment la geometria de la pantalla al SC_{cam}.

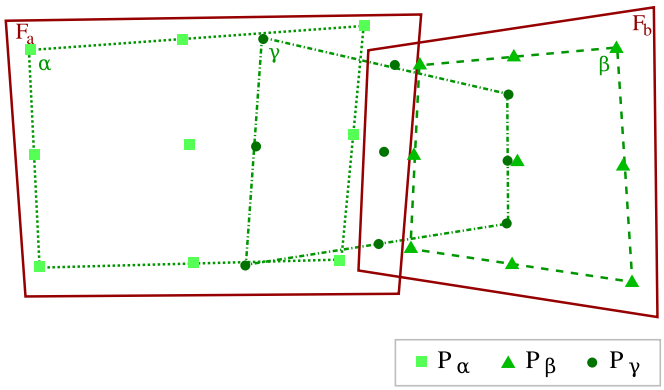


Figura 5.7: Exemple d'una paret amb $N = 3$ projeccions en verd i $M = 2$ captures en vermell.

Tenir M càmeres implica tenir M sistemes de coordenades diferents. Tenint en compte que l'objectiu és calcular les coordenades de la pantalla final, i que per a fer-ho necessitem tenir totes les àrees projectades en un únic sistema de coordenades, és prou clar que tenir la informació en M sistemes de coordenades diferents és un problema que cal resoldre.

Per tal d'aconseguir-ho les captures tenen una particularitat que permet solucionar el problema dels M sistemes de coordenades: les zones fotografiades no són disjuntes.

Siguin a i b les càmeres de què disposem i α, β i γ els projectors. Definim F_c com la captura que ha fet la càmera c, P_p com un dels punt del projector p i P_{pc} com un dels punts del projector p capturat per la càmera c.

Si ens fixem en l'exemple de la figura 5.7, les captures F_a i F_b tenen en comú que ambdues veuen punts P_γ, però que, en canvi, només a veu punts P_α i només b veu punts P_β. El problema, doncs, és que no tenim els tres conjunts de punts en el mateix sistema de coordenades: o bé ho estan P_α i P_γ, i P_β queda fora, o bé P_β i P_γ, i és P_α qui no hi està present. Tenint en compte que necessitem tenir-los tots representats al mateix espai, haurem de moure el tercer conjunt de punts (P_β en el primer cas, P_α en el segon) cap al sistema de coordenades dels altres dos.

Definim H_{pc} com l'homografia que transforma un punt en el sistema de coordenades del *frame buffer* de p a un altre punt en el sistema de coordenades de c, la qual s'ha calculat com hem descrit en aquest capítol. Per altra banda, també definirem H_{cp} com l'homografia que converteix un punt en el sistema de coordenades de la càmera c a un punt del *frame buffer* de p; és a dir, $H_{pc}^{-1} = H_{cp}$.

A l'exemple que estem tractant, les homografies de què disposem[2] són $H_{\alpha a}$, $H_{\gamma a}$, $H_{\gamma b}$ i $H_{\beta b}$. Si volem tenir tots els punts en el sistema de coordenades de, per exemple, a, només caldrà aplicar les homografies convenientment. Per a passar obtenir un punt P_α al sistema de coordenades d'a utilitzarem directament l'homografia $H_{\alpha a}$; per a fer-ho amb punts P_γ utilitzarem $H_{\gamma a}$; i, finalment, per a fer-ho amb els punts P_β haurem d'aplicar una composició de matrius:

[2]No tenim les homografies $H_{\alpha b}$ ni $H_{\beta a}$ perquè, evidentment, no s'han pogut calcular; la càmera a no veu els punts del projector β ni la càmera b els d'α.

El que volem és poder passar qualsevol punt P_β al sistema de coordenades d'a[3], però, com ja hem vist, no disposem de l'homografia $H_{\beta a}$. El que farem, doncs, és passar un P_β qualsevol al sistema de coordenades de b aplicant $H_{\beta b}$. Tot seguit, aplicant $H_{\gamma b}^{-1}$ passem aquest punt al sistema de coordenades del projector γ. Finalment, només ens cal aplicar $H_{\gamma a}$ a aquest nou punt per tal de tenir P_β al sistema de coordenades d'a. Formalment, tenim que:

$$H_{\beta a} = H_{\beta b} H_{b\gamma} H_{\gamma a}$$

La solució descrita és vàlida per a qualsevol nombre de projectors i càmeres, sent sempre possible transformar un punt qualsevol d'un projector donat al sistema de coordenades de qualsevol càmera. Ara bé, el fet d'anar concatenant les matrius implica un subtil problema, i és que es va *acumulant error*. Tal i com s'han construït les homografies de què es parteix, aquestes són susceptibles de tenir petits errors de precisió; a mesura que anem movent un punt d'un sistema de coordenades a un altre, tot aplicant-li les homografies, anem introduint-hi més i més error.

[CSWL02] introdueix el concepte *camera homography tree*, amb el qual s'aconsegueix una solució que minimitza l'error resultant. Tot seguit, es comenta breument la idea que hi ha al darrere.

5.6.1 *Camera homography trees* [CSWL02]

Sigui $G(V, E)$ un CHG (*Camera Homography Graph*, Graf d'homografies de càmera), on cada vèrtex del conjunt V és la captura d'una càmera i una aresta E correspon a una homografia directament computable entre dues vistes, o, altrament dit, una aresta connecta dos vèrtexs només si tenen, com a mínim, un projector en comú.

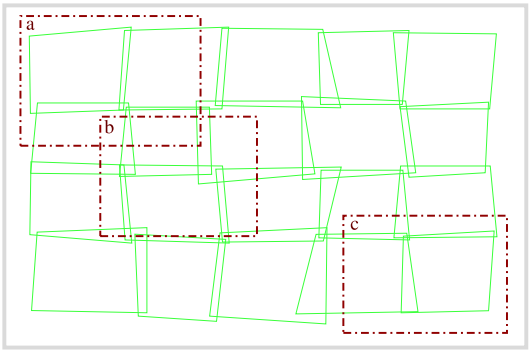
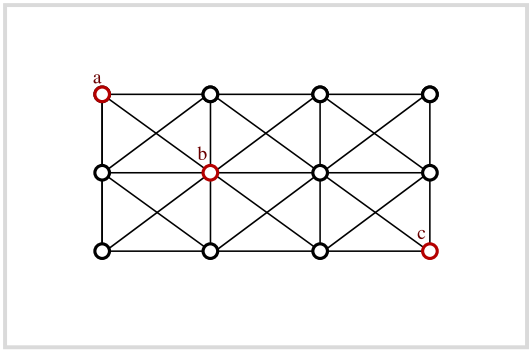


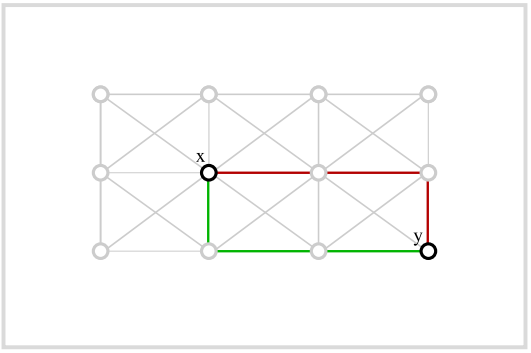
Figura 5.8: Àrees projectades en verd i tres captures en vermell de les, en aquest exemple, dotze realitzades per les càmeres.

Quan s'ha construït aquest graf (figura 5.9(a)), i tenint en compte que el CHG és connex, és possible calcular l'homografia entre dos vèrtexs qualsevol composant una cadena d'homografies que estigui en algun dels camins (vegeu la figura 5.9(b)). En teoria, un cicle qualsevol al CHG hauria de ser l'homografia identitat, atès que es parteix d'un vèrtex x i s'arriba al propi vèrtex x. Quan es dóna la propietat que per a qualsevol cicle del graf l'homografia retornada és la identitat, llavors es diu que tenim un *CHG consistent*. El problema és que en el cas d'aquest projecte, degut a petits errors en la captura i el processament d'homografies, això no és cert; tenim un *CHG inconsistent*. Això vol dir que l'homografia calculada entre dos vèrtexs depèn del camí escollit.

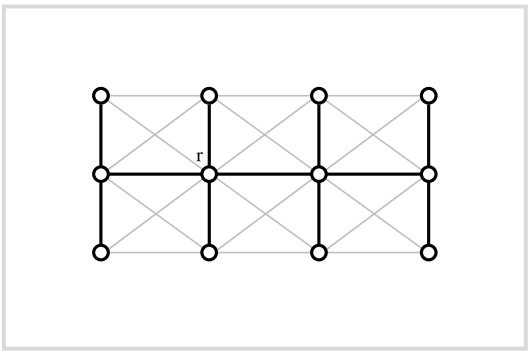
[3] És indiferent obtenir els punts P_β al sistema de coordenades d'a, que obtenir els P_α en b.



(a) Construcció del graf CHG.



(b) Exemple de camins entre dos vèrtexs x i y qualsevol.



(c) Arbre expandit a partir del vèrtex r.

Figura 5.9: Diagrama de la construcció d'un *camera homography tree* a partir d'un CHG.

Per definició, un *arbre* és un graf sense cicles. Conseqüentment, si el CHG obtingut és un arbre, aquest serà sempre consistent, atès que no hi haurà dos camins diferents entre dos vèrtexs.

L'objectiu és, doncs, construir un arbre a partir del graf inicial, que minimitzi la distància entre un vèrtex qualsevol i l'arrel de l'arbre, i que minimitzi la distància entre dos vèrtexs qualsevol[4].

Un algorisme concret que ho aconsegueix i que, a més a més, assoleix un nivell de precisió de *subpixel* està descrit a [CSWL02].

[4]Minimitzant la distància entre vèrtexs es minimitza la propagació de l'error.

Capítol 6

Xarxa

En aquest capítol es detallen quines arquitectures de xarxa hem fet servir per cadascuna de les parts. Cal tenir en compte que, com ja hem dit, el projecte consta de dues parts completament diferenciades i amb objectius diferents, i el disseny de la xarxa no n'és una excepció. Veurem que cada part s'ha dissenyat i implementat obeint als seus requeriments.

6.1 Objectius del disseny de la xarxa

A la primera part el que s'ha de fer són els calibratges, que és un procés relativament lent (de l'ordre de minuts). Per fer-les, hem de passar bastants dades d'un ordinador a l'altre, enviar ordres per fer accions, etc. Tota aquesta informació que passa a través de la xarxa és precisa, i no es poden admetre errors. Ja veiem que es necessita una xarxa amb control d'errors. Sabent també que els tipus de dades podran ser molt variats i que el temps no és una limitació, es pot fer una xarxa el més genèrica possible. En aquesta memòria, però, ens centrarem en explicar la xarxa de l'aplicació 3D.

Per la segona part, els objectius canvien. Tindrem una aplicació en 3D corrent en N ordinadors on s'ha de sincronitzar les càmeres a tots en temps real. També haurem de sincronitzar la geometria 3D, perquè com ja es veurà al capítol 9, aquesta es genera aleatòriament. Per tant, tenim dos objectius diferents:

comunicar a tots els ordinadors la geometria que han d'utilitzar

sincronitzar les càmeres el més ràpid possible.

Pel que fa als objectius de la xarxa física, es vol que sigui el més flexible possible. Quants menys cables, instal·lacions, etc. es necessitin millor, perquè, com hem descrit a la introducció, volem tenir mòduls independents, tots iguals, i idealment sense cap instal·lació entre ells.

6.2 Patrons utilitzats

A continuació veurem quins patrons de disseny de xarxa s'han utilitzat per complir els objectius anteriorment esmentats.

6.2.1 *màster - slave*

El primer que es pot observar és que pel que s'ha vist de moment, encara no tenim cap eina per decidir qui és de tots els ordinadors el que decideix quina geometria, o quina càmera hem

d'utilitzar. Veurem al capítol 9 que l'aplicació tindrà en un dels ordinadors, i només en un, un Wiimote associat i amb aquest calcularem la posició de la càmera virtual. De moment el que ens interessa per explicar la xarxa és el fet de que podem saber si hem de transmetre dades o hem d'esperar rebre-les. En canvi, sincronitzar la geometria és un problema una mica diferent. Com que cada ordinador al arrancar l'aplicació genera una geometria a l'atzar i totes són vàlides, es poden seguir molts criteris diferents per escollir la que tothom haurà de fer servir. En el cas que estem tractant, hem seguit l'aproximació que creiem més senzilla: qui envii les dades de la càmera, també enviarà la geometria.

Un cop se sap si hem de transmetre, s'ha de saber a qui. Un dels resultats que obtenim de la primera part és una llista d'adreces IP dels ordinadors que han participat al calibratge. Només haurem de comunicar aquests ordinadors i no d'altres, perquè si un ordinador no ha estat calibrat, no té sentit que utilitzi l'aplicació. Tampoc és estrictament necessari que hi siguin tots, podem començar a executar-la i a mesura que es vagin connectant ordinadors, es van sincronitzant.

Veient això, i amb els objectius que tenim, el patró màster - slave és sens dubte el que millor s'adequa als nostres objectius. L'arquitectura és molt simple, es tracta d'un màster i N esclaus. El màster sempre té les últimes dades i les transmet a tots els esclaus.

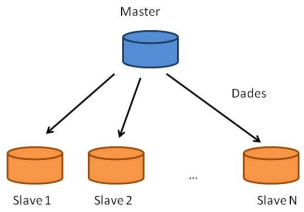


Figura 6.1: Master-slave.

En cas de voler detecció d'errors, els esclaus poden confirmar les dades, i en cas de voler la màxima velocitat per temps real, el màster assumirà que els missatges arriben bé sempre, i a més farà control del retard de xarxa. Aquest control ens servirà per evitar que el màster vagi més ràpid que els demés. Es farà un càlcul periòdic de la velocitat de la xarxa i el màster retardarà la seva execució en funció d'aquesta velocitat.

6.3 Implementació del patró

Fins aquest punt s'han descrit els objectius i el disseny de les xarxes. Comentarem ara detalls de la implementació.

6.3.1 *real time master - slave*

El patró de master - slave per enviar les dades de la càmera té com a requeriments ser el més ràpid possible, i no cal controlar errors. Aquestes dues condicions indiquen clarament que hem

d'utilitzar el protocol UDP. Tenim una altra condició, han d'estar el màxim de sincronitzats possible. Per resoldre aquest problema, bàsicament s'envien des del màster 3 datagrames UDP a tots els esclaus, s'esperen les seves respostes i es calcula el temps que ha passat. D'aquest temps se'n diu *round trip time*, o RTT, que és el temps que triga un datagrama a anar i tornar en una xarxa (vegeu figura 6.2). També ens referirem a aquest temps com retard de xarxa.

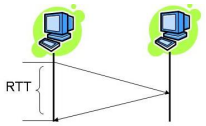


Figura 6.2: Round Trip Time.

Si el retard de xarxa és massa gran, com que el master és qui té primer les dades, si les consumeix immediatament es notarà que els ordinadors no estan sincronitzats. Aquest problema es resol agafant el *round trip time* que hem calculat, dividir-lo per dos i retardar el master. El procés és el següent: el master envia les dades, s'espera mig *round trip time* i les consumeix, i els esclaus triguen mig RTT en rebre les dades i les consumeixen directament. D'aquesta manera, tant el master com els esclaus triguen mig RTT a consumir les dades. Cal tenir en compte que el temps de retard de la xarxa és totalment dinàmic, perquè depèn de condicions canviables i externes al sistema, i per tant s'ha d'anar recalculant periòdicament. S'ha decidit fer el càlcul cada tres segons per evitar saturar la xarxa amb missatges de control i minimitzar l'impacte en la latència.

Pel que fa a les connexions, els esclaus obren un port UDP i l'escolten. Quan reben dades per aquell port, l'aplicació les rep i les decodifica. Com hem dit abans, tots els ordinadors tenen la llista de IP de tothom que ha intervingut al calibratge, per tant el master intentarà connectar-se a totes les IP de la llista. Si alguna no està disponible, s'esperarà en un thread apart. El master també obrirà un port per escoltar les respostes dels pings dels esclaus.

No hem implementat cap tipus de control de seguretat de xarxa. Queda com a feina futura, i es detalla a l'apartat 12.1.

6.3.2 *master - slave*

Aquest patró Master-slave té uns objectius diferents que l'anterior. No té restriccions de temps i es vol assegurar que les dades arriben. La millor opció és utilitzar TCP. El procés de connexió és el mateix, el master s'intenta connectar a totes les adreces IP de la llista, i si no estan disponibles, les espera en un thread apart. El master troba primer qui hi ha connectat en aquell moment, i envia la geometria a tothom. Si algú es connecta després, l'usuari haurà de seleccionar manualment quan enviar la geometria. S'ha decidit de fer així per respectar la simplicitat de l'arquitectura de tota l'aplicació.

Com fem servir TCP, a nivell d'aplicació no ens hem de preocupar del control d'errors, això es fa al nivell de transport. Si no hi ha errors, el procés és tan simple com a la figura 6.3, el master envia les dades i el nivell de transport dels slaves s'encarrega de confirmar-les.

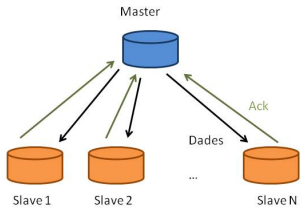


Figura 6.3: Master-slave implementat amb TCP.

Si ha algun error, com a la figura6.4, el slave no rep res, i per tant no torna un *Ack* al master.

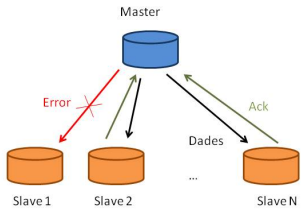


Figura 6.4: error a la xarxa, un slave no rep les dades.

Quan passa un cert temps sense que el master rebi l'*Ack* de tots els slaves, el nivell de transport del master retransmet les dades a aquells que no han enviat *Ack*.

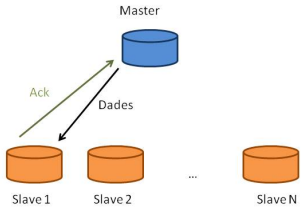


Figura 6.5: El master retransmet les dades que s'han perdut.

6.3.3 La xarxa física

La xarxa física que menys cables i instal·lació necessita és una xarxa *WiFi*. Les xarxes *WiFi* es poden configurar amb *router* o ad hoc. En el nostre cas, com l'únic requisit és que hi hagi connectivitat entre ells, podem escollir qualsevol dels modes. Per tenir el mínim possible d'infraestructura, recomanem el mode ad hoc, que al no necessitar router, permet que el sistema funcioni amb mòduls clònics i sense cap altre element extern als mòduls. Per més detalls, i per saber com es configura, ens remetem a la memòria del nostre company.

Capítol 7

Part 1: calibratge

Com ja hem dit a la introducció, aquest és un projecte fet per dues persones. Per aquest motiu, en aquest capítol farem constantment referències a la memòria del nostre company, sobretot en la part d'especificació i disseny.

7.1 Introducció i objectius

Abans d'entrar en detalls, aclarim quins dels objectius descrits a la introducció estarem acomplint amb aquesta part. Aquests són: el *calibratge automàtic* i també la *portabilitat*. Es vol implementar un sistema que, donada una disposició qualsevol dels projectors envers una paret plana (i sempre que aquests dos tinguin una àrea en comú), es pugui calcular automàticament una pantalla el més gran possible al seu interior.

També és interessant desenvolupar una eina que permeti que l'usuari corregeixi les mides de la pantalla resultant per adequar-les al espai on s'utilitzaran. D'aquesta manera s'aconsegueix l'objectiu de *portabilitat* en el sentit que es crea una independència envers on s'utilitzarà l'aplicació

Per tant, els dos punts clau són:

- Calibratge automàtic.
- Eina per a corregir la pantalla resultant.

Per últim, hem de crear un format per tal que l'*aplicació d'usuari* que es descriu al capítol 9 pugui utilitzar els resultats obtinguts després del calibratge. Afegim, doncs, el punt:

- Exportació dels resultats del calibratge cap a l'aplicació d'usuari.

7.2 Especificació

En aquest capítol només es mostren les parts més importants de l'especificació, perquè aquesta feina la ha fet, majoritàriament, el nostre company, David Aguilera.

7.2.1 Restriccions

Saber quines restriccions té una aplicació és vital per entendre quin ús se'n pot fer. Tot seguit aclarirem què suposem com a cert en tot moment i què no podem esperar que faci el sistema.

Supòsits

En primera instància, cal posar de manifest aquelles parts del sistema que s'assumeixen que són certes i que, per tant, no ens han de preocupar:

- Un mòdul projector - càmera disposa de, com a mínim, una càmera.
- El sistema operatiu sobre el que s'executarà la nostra aplicació és *Microsoft Windows XP*.
- Els ordinadors disposen de xarxa sense fils, i es pot configurar com s'especifica al capítol 6.
- La paret sobre la qual es realitza el calibratge és completament plana i blanca.

Què no fa el sistema

Per tal de delimitar bé l'àmbit d'ús del sistema de calibratge, indiquem què és el que no pot fer:

- El sistema no pot calibrar els projectors si aquests projecten a més d'una paret.
- El nombre màxim de mòduls projector - càmera que es pot enregistrar al sistema és de dos.
- El sistema només corregeix les intensitats dels projectors, però no pot "dissimular" el fons sobre el qual s'està projectant (de fet, ja hem dit que es projecta contra una paret blanca).
- El sistema no calcula automàticament una pantalla que estigui alineada amb els eixos vertical i horitzontal del món real.
- El sistema no calcula la pantalla d'àrea més gran continguda a la zona projectada.

7.2.2 Casos d'ús

En aquest apartat es mostren només els casos d'ús més importants. La resta els podrem trobar a la memòria del nostre company.

1. Exportar els resultats del procés de calibratge	
Descripció	
El subsistema de calibratge ha de poder comunicar-se amb l'aplicació d'usuari (veure capítol 9) i, per tant, tots els resultats que es computin han de ser accessibles per al segon.	
Actor principal	Usuari
Precondició	
S'han realitzat correctament els calibratges geomètric i cromàtic.	
Criteri de validació	
S'han desat els resultats que espera com a entrada l'aplicació d'usuari correctament, segons el format establert.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari decideix que vol iniciar l'aplicació d'usuari.	

(continua a la següent pàgina)

(ve de la pàgina anterior) 2. L'usuari indica al sistema que vol exportar els resultats del calibratge al format que necessita l'aplicació.	
	3. El sistema desa els resultats en el format pactat a la ubicació especificada a la seva configuració.
Cursos alternatius	

---

2. Calibrar geomètricament el sistema

Descripció

El sistema ha de poder calibrar la geometria dels projectors, estan aquests ubicats físicament de forma arbitrària, per tal d'aconseguir calcular una pantalla virtual que hi estigui continguda.

Actor principal Administrador

Precondició

- Existeix una àrea de solapament entre les zones projectades per ambdós projectors.
- Hi ha algun mòdul registrat al sistema.

Criteri de validació

El sistema coneix la disposició dels projectors i ha calculat les homografies[1] necessàries per a poder dibuixar correctament a l'interior de la pantalla.

Curs típic d'esdeveniments

Usuari	Sistema
1. L'administrador decideix que vol corregir l'alineació dels projectors. 2. L'administrador indica al sistema que iniciï el procés de calibratge geomètric.	
	3. El sistema inicia el procés de calibratge. 4. El sistema computa els resultats. 5. El sistema pregunta a l'usuari on vol desar els resultats que ha obtingut.
6. L'administrador indica la ubicació on vol que es desin aquests resultats.	
	7. El sistema desa els resultats.

Cursos alternatius

---

[1] En aquesta fase d'especificació del projecte no acostuma a ser bona idea parlar de solucions tecnològiques a un problema donat (com gestionar la correcció geomètrica dels projectors, en aquest cas). De totes maneres, tenint en compte que l'ús de les homografies és una imposició inicial del projecte i que, per tant, no es contempla cap altra solució, considerem vàlid mencionar-les.

3. Calibrar cromàticament el sistema

Descripció

El sistema ha de corregir les intensitats dels projectors, per tal d'evitar que un projector brilli més que l'altre o que la zona de solapament d'ambdós projectors sigui perceptible.

Actor principal Administrador

Precondició

- El sistema disposa d'un calibratge geomètric vàlid per a la disposició actual dels projectors (computat o carregat).
- El nombre de mòduls registrats al sistema es corresponen a aquest calibratge geomètric.

Criteri de validació

El sistema corregeix la intensitat dels projectors, fent que resulti impossible percebre que la pantalla calculada estigui formada per més d'un projector.

Curs típic d'esdeveniments

Usuari	Sistema
1. L'administrador decideix que vol corregir les intensitats de llum que els projectors aporten a la pantalla calculada.	
2. L'administrador indica al sistema que iniciï el procés de calibratge cromàtic.	
	3. El sistema inicia el procés de calibratge.
	4. El sistema computa els resultats.
	5. El sistema pregunta a l'usuari on vol desar els resultats que ha obtingut.
6. L'administrador indica la ubicació on vol que es desin aquests resultats.	
	7. El sistema desa els resultats.

Cursos alternatius

4. Definir dimensions finals de la pantalla

Descripció

L'usuari ha de poder corregir la geometria de la pantalla que el sistema ha computat automàticament, per tal d'adaptar-la a les seves necessitats[2].

Actor principal Usuari

Precondició

- El sistema disposa d'un calibratge geomètric vàlid per a la disposició actual dels projectors (computat o carregat).
- El nombre de mòduls registrats al sistema es corresponen a aquest calibratge geomètric.

Criteri de validació

El sistema ha computat unes noves homografies per tal d'adaptar les imatges a la nova geometria indicada per l'usuari de la pantalla.

(continua a la següent pàgina)

[2]En general, aquesta correcció manual que aplica l'usuari vol dir alinear la pantalla als eixos vertical i horitzontal de la realitat

(ve de la pàgina anterior)

Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari decideix que vol canviar la geometria actual de la pantalla continguda a l'àrea de projecció dels projectors. 2. L'usuari indica al sistema que vol canviar-la.	
	3. El sistema pregunta a l'usuari quines noves coordenades vol per a cadascun dels quatre punts que conformen la pantalla.
4. L'usuari introdueix les noves coordenades.	
	5. El sistema computa les noves homografies segons la informació que li ha donat l'usuari. 6. El sistema pregunta a l'usuari on desar els resultats.
7. L'usuari indica la ubicació on vol que el sistema els desi.	
	8. El sistema desa els resultats a la ubicació explicitada.
Cursos alternatius	

---

5. Comprovar els resultats del calibratge

Descripció

El sistema ofereix una eina a l'usuari que li permeti comprovar quin és el resultat que s'obtindria si es dibuixés una imatge qualsevol a la pantalla que s'ha obtingut de l'alineament geomètric (i cromàtic, si s'hagués dut a terme també).

Actor principal Usuari

Precondició

- El sistema disposa d'un calibratge geomètric vàlid per a la disposició actual dels projectors (computat o carregat).
- El nombre de mòduls registrats al sistema es corresponen a aquest calibratge geomètric.

Criteri de validació

S'ha enviat a pintar a la pantalla computada pel sistema de calibratge la imatge que l'usuari vol per a comprovar els resultats obtinguts.

Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari vol comprovar quin resultat s'obté amb el calibratge que ha donat el sistema. 2. L'usuari indica al sistema quina imatge vol utilitzar per a dur a terme el test.	
	3. El sistema carrega la imatge i la projecta convenientment per a què estigui continguda a la pantalla "virtual" computada.
Cursos alternatius	

7.2.3 Diagrama de classes

En aquest apartat es descriu el diagrama de classes que dóna suport a tot el sistema. Per veure més detalls i entendre millor cadascuna de les parts, ens remetem, un cop més, a la memòria de David Aguilera.

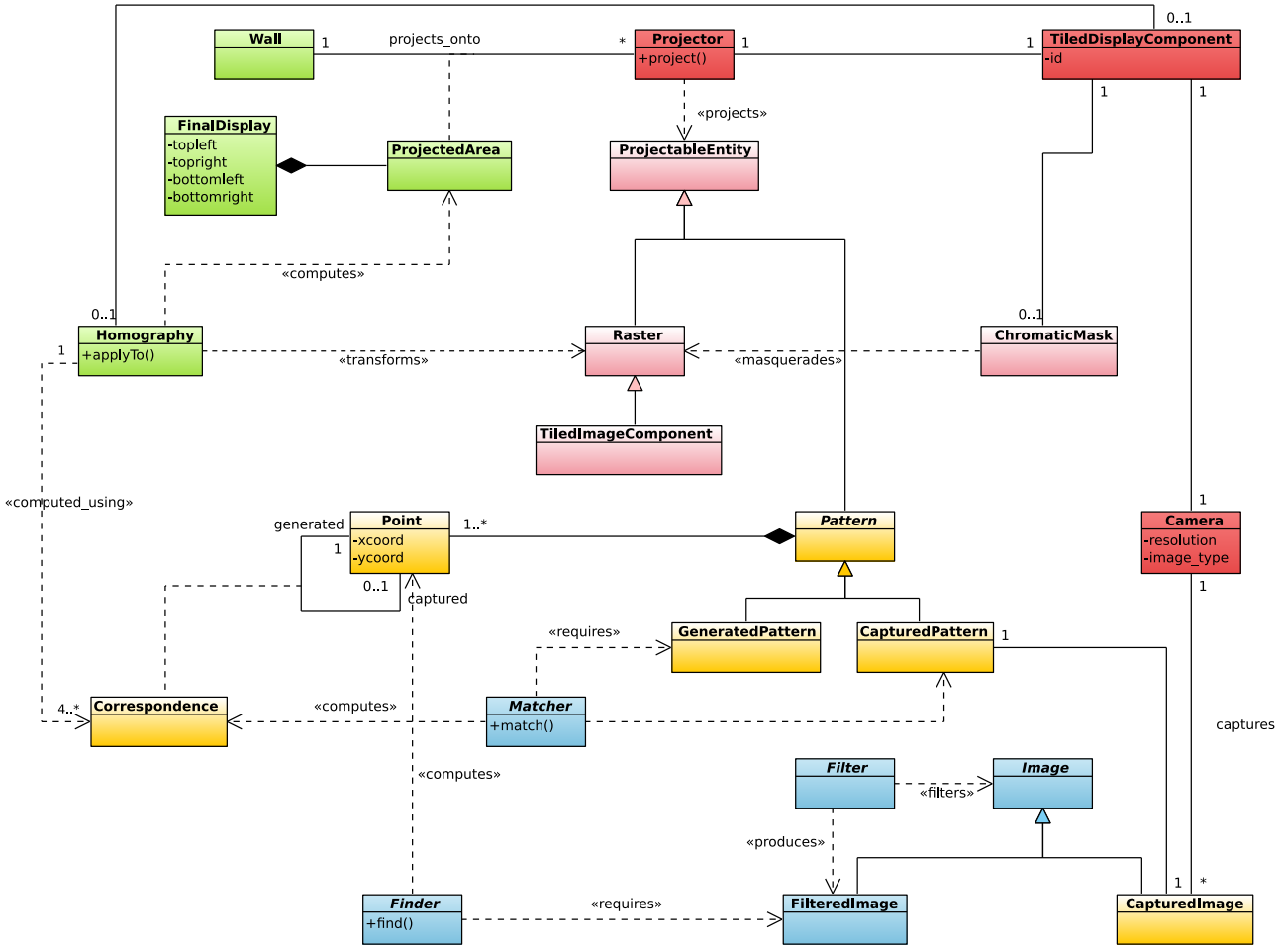


Figura 7.1: Diagrama de classes d'especificació del mòdul de calibratge.

7.3 Disseny

Com a l'apartat anterior, només es comenten els apartats més importants, i pels detalls i explicacions més extenses caldrà llegir la memòria del nostre company.

7.3.1 Arquitectura del sistema

L'arquitectura del sistema és una descripció dels diferents mòduls que el formen i de les relacions entre ells. El que farem és determinar l'organització final del sistema de calibratge, tot aplicant *patrons de disseny*:

Patró de disseny és una solució genèrica a un problema conegut i comú al disseny de programari. No és quelcom traduïble directament a codi font, però sí que dóna una plantilla o descripció de com solucionar el problema en diferents situacions.

Arquitectura en tres capes

Una de les arquitectures més comuns a l'hora de desenvolupar una aplicació és l'*arquitectura en capes*, la qual agrupa els components del sistema en diferents *capes*, de tal manera que cadascun d'ells només pot comunicar-se amb altres components de la mateixa capa o les contigües.

És molt habitual trobar sistemes que segueixen aquesta arquitectura, amb un model de tres capes: la de *presentació*, on hi ha els components que interaccionen amb l'usuari; la de *domini*, on hi ha tota la lògica del sistema, encarregada de controlar la validesa de la informació que gestiona, ordenar l'execució d'accions, etc.; i la de *gestió de dades*, la qual procura que tota la informació que gestiona el nostre sistema i hagi de ser persistent ho sigui.

Arquitectura distribuïda

Per altra banda, hem comentat que el nostre sistema ha de ser, forçosament, distribuït, atès que és un dels objectius i requisits que tenim. Per tant, des del punt de vista de com organitzar el sistema de forma distribuïda, sí que s'ha seguit una arquitectura concreta.

D'entre les diferents arquitectures distribuïdes que podem aplicar, nosaltres hem decidit utilitzar la que ens ofereix el patró *master - worker*, descrit al capítol 6.

Si tenim en compte que el calibratge dels projectors és un procés que ha d'estar correctament sincronitzat, atès que a cada instant de temps només es poden estar enviant patrons per un únic mòdul i fer que tots el capturin, es pot veure que el patró *master - worker* s'adapta molt bé a les nostres necessitats: un dels mòduls actuaria com a *master* i la resta (i també el propi *master*) serien els *workers*.

7.3.2 Diagrama de classes

El diagrama de classes que resulta del procés de disseny és el de la figura 7.1. De nou, ens remetem a la memòria del nostre company per més detalls i explicacions més concretes de les diferents parts.

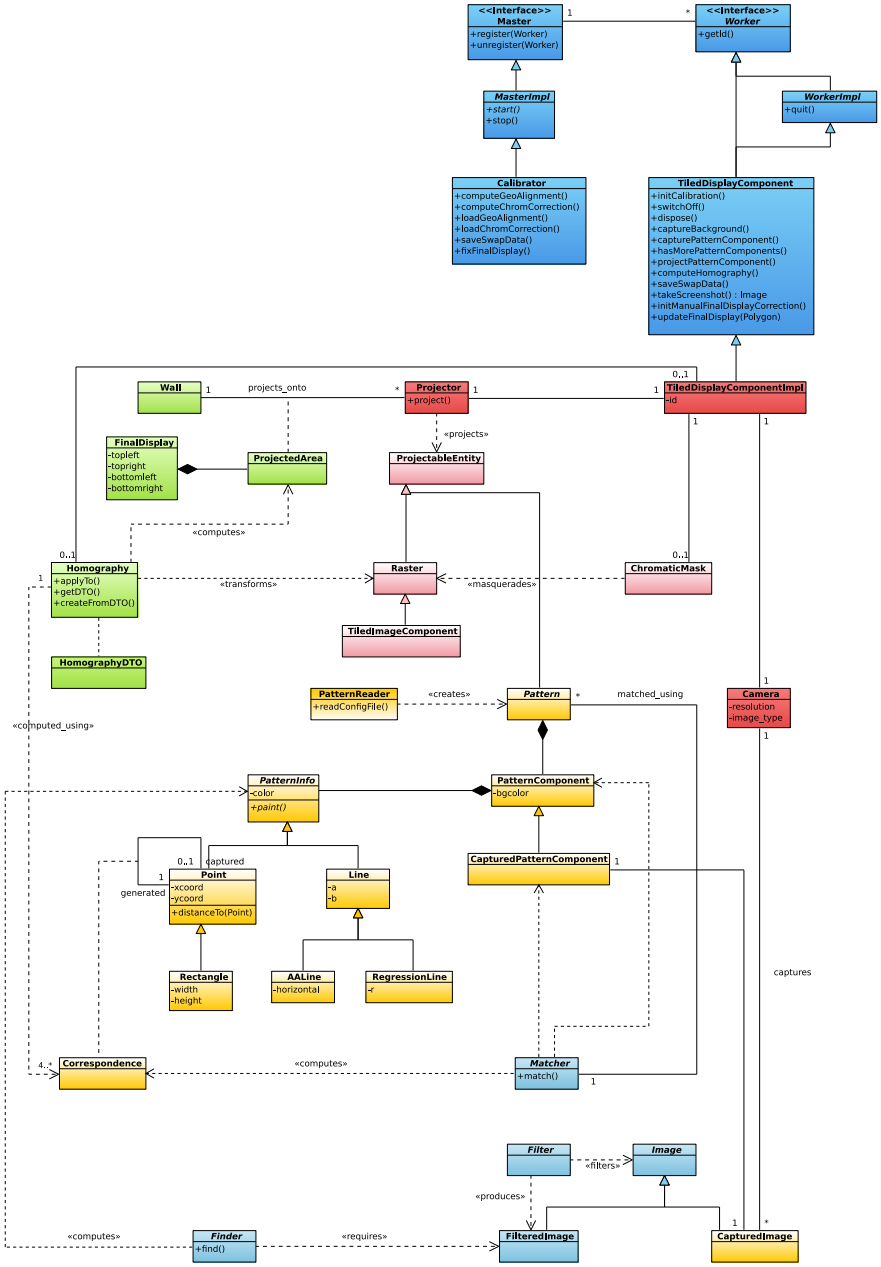


Figura 7.2: Diagrama de classes de disseny del mòdul de calibratge.

7.3.3 Gestió dels resultats

Alguns dels requeriments que tenim indiquen com treballa el sistema amb els resultats de què disposem. En concret, defineixen que aquests s'han de poder *desar* i *recuperar*, així com *exportar* al format que necessita l'aplicació d'usuari.

Exportar resultats per a l'aplicació d'usuari

Per a exportar els resultats del procés de calibratge també utilitzem fitxers. Ambdós sistemes (*calibratge* i *aplicació d'usuari*) tenen accés al sistema de fitxers del sistema operatiu i no requereixen cap mena de configuració addicional per treballar-hi, justificant, així, la utilització d'aquesta solució.

La ubicació d'aquests *fitxers d'intercanvi* ha de ser coneguda per ambdues aplicacions, atès que el que una desi a un lloc, l'altra haurà de poder-ho llegir. Per tal de coordinar-les, es defineix un nou fitxer de configuració que indica quines són aquestes ubicacions.

7.4 Implementació

El procés d'implementació no sempre és el més complex, ja que normalment el disseny clarifica la implementació de manera que es pugui desenvolupar de manera quasi automàtica si es compta amb programadors amb l'experiència necessària.

El nostre gran repte era superar les dificultats de programar amb entorns totalment desconeguts, i inclús desenvolupar algoritmes que no havíem previst que fossin tan complexes. Més endavant en descrivim alguns, i per veure'n més, ens remetem un altre cop a la memòria del nostre company.

7.4.1 Utilització de la càmera

Per tal de poder utilitzar la càmera des de Java i disposar de la màxima resolució i qualitat que ofereix, hem utilitzat la llibreria *DirectShow*. Per a més informació sobre com s'ha integrat al nostre projecte, vegeu l'apèndix A.2.

7.4.2 Implementació d'algorismes (imatges)

La implementació dels algorismes de visió per computador (corresponents a les classes `Filter`, `Matcher` i `Finder`) estan comentats de forma exhaustiva al propi codi.

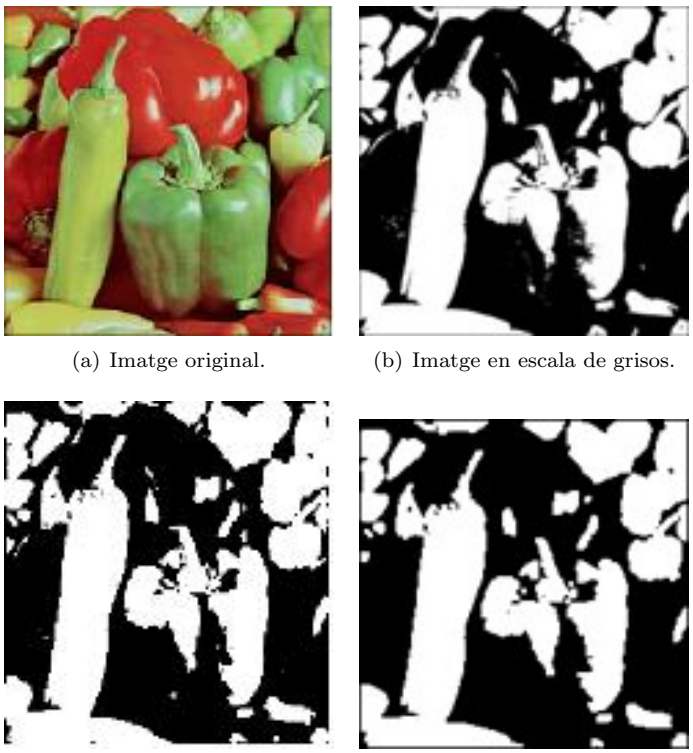
De totes maneres, a continuació explicarem amb una mica més de detall i sense haver d'anar a nivell de codi en què consisteixen aquests algoritmes.

Filters

Els `Filter` són algoritmes de manipulació d'imatges per extreure dades o per eliminar soroll de les imatges que rebem. Un exemple és el `BasicImageFilter`, que el que fa és passar la imatge a escala de grisos, equalitzar-la, binaritzar-la i aplicar-hi un filtre d'obertura, o *opening*.

La raó per la qual s'acaba binaritzant les imatges sempre, és per poder tractar amb les dades d'una manera molt més ràpida i àgil, perquè amb una comparació tan ràpida com veure si un pixel val 1 o 0, ja es pot saber si aquella dada és interessant o no. A més, algoritmes com contar punts en una àrea, o filtres com el d'obertura serien considerablement més complicats d'implementar i clarament menys eficients. El filtre *opening* és un dels filtres més clàssics i més utilitzats per eliminar soroll. Es tracta d'aplicar a la imatge consecutivament l'operador morfològic d'erosió i de dilatació. Aquests operadors recorren tots els pixels de la imatge i comparen el pixel observat amb els del seu voltant, i depenent dels valors dels quals està rodejat i de la matriu morfològica que s'utilitza, el pixel canvia o no de valor. Per saber-ne més sobre algoritmes típics de visió per computador, vegeu l'apèndix E.

L'algoritme que més s'utilitza per filtrar les captures de patrons de línies és el següent: es projecta la pantalla en negre, i es captura. Aquesta imatge és el fons, i s'utilitza per restar-la



(a) Imatge original. (b) Imatge en escala de grisos.

(c) Imatge binaritzada. (d) Imatge amb el filtre de *opening*.

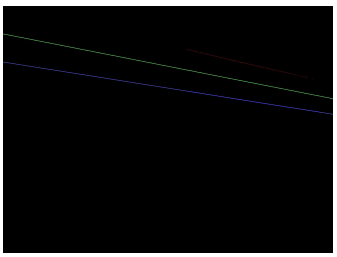
Figura 7.3: Procés de filtrat d'una captura.

a les captures amb patrons, i així ressaltar-los. Un cop fet aquest pre-procés, se li aplica un `BasicImageFilter`. Després es recorren tots els pixels que han quedat a 1 d'aquesta. Cadascun d'aquests pixels correspon a una coordenada d'una recta que hem capturat, i com que hem enviat vàries línies alhora, hem de determinar de quina es tracta. Per això cada línia del patró és d'un color diferent. Abans de binaritzar es disposava la informació dels colors, per tant pixel a 1 de la imatge filtrada correspon a un color el qual es pot decidir quin és fàcilment: agafem el color predominant, i per predominant ens referim a que la seva intensitat sigui més gran que la dels altres colors en un percentatge llindar d'entrada.

Finders

Els `Finder` són cercadors d'elements que sabem que es troben en les captures. Per exemple, el `BlobsFinder` escaneja la imatge en busca de grups de punts junts. Si nosaltres enviem a pantalla figures geomètriques plenes i les capturem amb la càmera, segurament siguem capaços de detectar-les. Ara bé, si no filtrem la imatge que ens arriba, confondrem el soroll que ens arriba de defectes a la càmera, o petits canvis d'il·luminació. El filtre més important, i el que es fa servir més, és el `RegressionLinesFinder`. Aquest passa la imatge per un filtre que separa la imatge per colors *R, G i B*i amb la imatge filtrada busca una rectes per cada color. Per calcular les rectes, agafem el conjunt de punts de cada color i calculem la recta de regressió. Si la dispersió és massa gran, descartem els punts que s'allunyen massa de la recta i tornem a fer el procés. La dispersió màxima que acceptem és un paràmetre de l'algoritme descrit en els fitxers de configuració. Vegeu les figures 7.4, tretes d'una execució del nostre sistema.

(a) Imatge filtrada que rep el `Finder`.



(b) Imatge amb les línies trobades. El càlcul de les equacions de les línies verda i blava ha tingut èxit, i a la imatge superposem les rectes calculades. La línia vermella no l'-ha pogut processar

Figura 7.4: Diferents exemples de dispositius *haptics*.

Matchers

Un `Matcher` busca les correspondències entre les figures enviades per pantalla i les rebudes per les captures de la càmera. Utilitzem els filtres i els cercadors descrits anteriorment. Menció especial mereix el `RegressionLinesMatcher`, que treballa d'una manera una mica diferent. Nosaltres enviem a pantalla línies horitzontals i verticals i calculem les interseccions, que donen un conjunt de punts E. De les captures també calculem les interseccions i se'n treu un conjunt C. Per relacionar aquests conjunts, relacionem cada línia capturada amb la original i ja podem fer les correspondències. El procés d'associar cada línia capturada amb la que s'envia és molt senzill. Es guarda en un vector cada captura segons l'ordre en que s'ha fet, igual que es guarda cada patró segons l'ordre en que s'ha projectat. Cada patró i cada captura consten de tres línies, i amb el `RegressionLinesFinder` som capaços de discernir cada línia de la captura segons el color. A partir d'aquesta informació, associar les línies projectades amb les capturades és trivial. Els patrons de línies també tenen la informació de si són verticals o horitzontals, per tant, al calcular les interseccions de les rectes calculades a partir de les captures hem de tenir en compte que per cada recta horitzontal només s'ha de intersecar amb les rectes verticals i a la inversa.

7.5 Integració i proves

L'última etapa per concloure qualsevol sistema software són les proves. Aquesta fase és tant important com la resta per detectar els errors comesos al llarg del desenvolupament i poder afirmar que un sistema és estable i satisfà els requisits i les restriccions que s'especifiquen anteriorment.

Un dels mètodes per comprovar-ho és realitzar les *proves unitàries* dels diferents components clau que composen el nostre sistema, per tal d'assegurar que, de forma independent, funcionen correctament. Un cop es pot afirmar que cada part funciona correctament, es van integrant components i comprovant que aquestes integracions produeixen els resultats esperats

En aquest capítol es comenten els punts més importants de les proves unitàries i d'integració que hem dut a terme, especialment en l'apartat de visió per computador. Per la resta de proves unitàries, vegeu la memòria del nostre company.

7.5.1 Proves unitàries

Les proves unitàries són un mètode emprat per a verificar que les diferents unitats de codi funcionen correctament. Una unitat és la part més petita de la qual se'n pot comprovar el funcionament.

Tot seguit, anem a comentar breument les proves més importants que s'han dut a terme de forma unitària:

Correspondències de punts

L'objectiu final del mòdul és el còmput de les homografies, i per tant, es necessiten un mínim de quatre correspondències entre punts que s'envien i punts que es capturen. S'ha verificat amb imatges generades manualment que els sistemes de manipulació d'imatges descrits a l'apartat 7.4.2 troben els punts correctament, i després ho hem integrat amb les imatges rebudes per la càmera.

Filtres d'imatges

S'ha comprovat que tots els filtres d'imatges donaven el resultat esperat. Cada filtre té uns resultats esperats diferents, per exemple alguns només separen colors, altres eliminen soroll, etc. Primer s'han generat imatges sintètiques per provar els algoritmes i després els hem integrat amb les captures de la càmera.

Captura d'imatges

Com es pot veure a l'apèndix A.2, la llibreria *DirectShow* té alguns inconvenients. S'ha hagut de comprovar que les captures que rebem fossin les adequades, i no hi hagués una barreja de patrons. Hem trobat un interval de temps que ens assegura tenir la imatge desitjada sense haver de carregar massa en temps tot el calibratge.

Capítol 8

Seguiment de l'usuari

En aquest capítol s'explica amb detall en què consisteix el *head tracking* i perquè utilitzar-lo representa una millora qualitativa molt gran pel que fa al nivell d'immersió que s'obté.

També es descriu la manera en què s'implementa i integra al nostre sistema.

8.1 Què és el *head tracking*?

La traducció literal de *head tracking* vol dir *seguiment del cap*. Es tracta d'una tècnica que permet a l'ordinador conèixer la posició del cap de l'usuari respecte a alguna posició concreta. En el nostre cas, ens interessa conèixer-la respecte la pantalla on es visualitza l'entorn 3D.

Disposar d'aquesta informació permet millorar la sensació d'immersió que el sistema ofereix a l'usuari, atès que allò que se li mostra és congruent amb la seva posició. Això vol dir que l'entorn virtual respon de manera natural als moviments que fa l'usuari, oferint-li informació addicional sobre la forma i dimensions d'un objecte tridimensional.

La figura 8.1 mostra la idea que hi ha darrere del seguiment de l'usuari.

Si es parteix de la imatge 8.1(a), la qual se suposa s'està observant des del davant, i l'usuari es desplaça lleugerament cap a un costat, el que veurà quan no hi ha el *head tracking* activat és el que es veu a la imatge 8.1(b). En canvi, si s'activa, el sistema serà conscient de què l'usuari està mirant la pantalla des d'una nova perspectiva. La figura 8.1(c) mostra el perfil de la cara, doncs és el que l'usuari esperaria veure des de la seva nova ubicació.

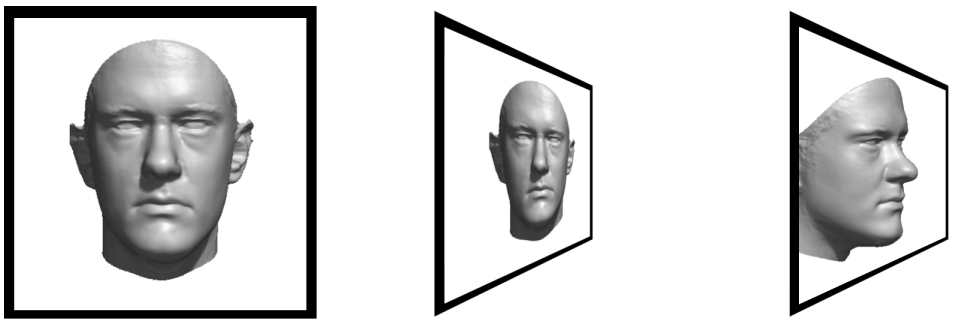
8.2 Videoconsola *Wii*® (*Nintendo*®)

El comandament de la consola *Nintendo*® *Wii* (*Wiimote*®) consta, entre d'altres sensors, d'una càmera d'infrarojos. Aquesta càmera té un angle d'obertura aproximada de 45º i una resolució de 1024×768 *pixels*. També consta d'un transmissor *Bluetooth* per a comunicar les dades.

La idea per a realitzar el *head tracking* amb el *Wiimote*®[1] és fer que controli en tot moment la posició de dos punts infrarojos, els quals suposarem associats a la posició del cap de l'usuari. A partir d'una posició inicial donada, per la qual s'assumirà que l'usuari està mirant la pantalla des del centre, calcularem la posició relativa del cap respecte la pantalla quan aquests es desplacin.

Quan hom està jugant a la *Wii*®, el que fa és controlar el *Wiimote*® i se situen els LED (*Light Emisor Diode*, Díodes emissors de llum) infrarojos sota la pantalla. En el nostre cas,

[1] Vegeu [CL].



(a) Imatge d'una cara en primer pla, vista des del davant.

(b) Imatge de la cara vista des del lateral, sense el *head tracking* activat.

(c) Imatge de la cara vista des del lateral, amb el *head tracking* habilitat.

Figura 8.1: Exemple de *head tracking* (imatges obtingudes de [BV99]).

però, la configuració serà inversa: el *Wiimote*® prop de la pantalla, enfocant cap a l'usuari, i en el cap d'aquest hi posarem dos LED infrarojos.

8.3 Integració amb el projecte

Per tal d'integrar el *head tracking* al projecte cal resoldre, principalment, dos problemes:

- Comunicació entre l'aplicació d'usuari i el *Wiimote*®.
- Gestió i càlculs de la informació que ens proporciona el *Wiimote*®.

Per tal de comunicar el comandament i l'ordinador s'ha utilitzat la *llibreria WiimoteLib*, amb la qual ja s'han realitzat diversos projectes. Es tracta d'una *llibreria* molt, que fins i tot és capaç de gestionar més d'un comandament alhora.

Finalment, només cal veure com processar la informació que s'obté, cosa que s'explica al següent subapartat.

8.3.1 Càlculs

Per tal de fer el seguiment d'un objecte a l'espai (la posició del cap) no n'hi ha prou en disposar de només dos punts. Per a poder fer-ho d'aquesta manera s'han d'assumir una sèrie de simplificacions i s'ha de ser conscient de què el que s'obté és només una aproximació a la ubicació real.

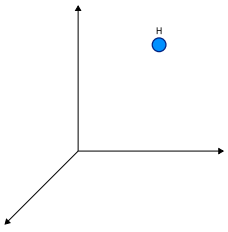


Figura 8.2: Posició del cap.

L'objectiu d'aquest apartat és, doncs, trobar una fórmula per a obtenir la posició Hx, Hy, Hz, tenint com a dades la distància entre els dos LED i la imatge que rebem des del *Wiimote®*.

Figura 8.3: Imatge de la càmera del *Wiimote®*.

Sabent que estem en un espai com a la figura 8.2, centrat a la càmera del *Wiimote®*, i que aquesta ens dóna una imatge com la de la figura 8.3, definim WP_1 i WP_2 com els dos punts que veiem des del *Wiimote®* i *pointDist* com la distància en el pla entre ells (vegeu la figura 8.4):

$$pointDist = \sqrt{(WP1_x - WP2_x)^2 + (WP1_y - WP2_y)^2}$$

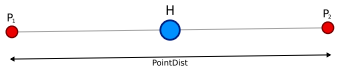


Figura 8.4: Perspectiva des de la càmera del *Wiimote®*.

Si tenim en compte que la càmera té una resolució horitzontal de 1024 *pixels* i una obertura de 45º, podem deduir l'angle que hi ha entre dos *pixels* consecutius tal i com es descriu a l'equació 8.1.

$$radiansPerPixel = \frac{\pi/4}{1024} \tag{8.1}$$

En primer lloc, calculem l'angle α del triangle rectangle que formen W, P_2 i H amb 8.2 (vegeu la figura 8.5). Recordem que estem suposant que l'usuari sempre mira cap a la pantalla i que, per tant, el triangle serà sempre rectangle.

$$\alpha = radiansPerPixel \cdot \frac{pointDist}{2} \tag{8.2}$$

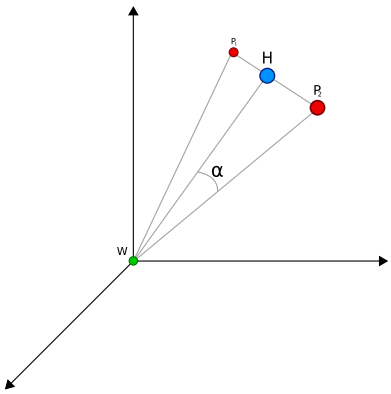


Figura 8.5: Angle α.

A partir d'aquest angle α podem calcular la distància entre la càmera i H com:

$$H_z = \frac{dotDistance/2}{\tan(\alpha)/screenHeight}$$

Ara calculem les coordenades del punt mig entre WP_1 i WP_2.

$$avgX = \frac{WPx_1 + WPx_2}{2}$$

$$avgY = \frac{WPy_1 + WPy_2}{2}$$

Aquest punt $(avgX, avgY)$ ens serveix per a situar les coordenades (x, y) d'H. Calculant quants *pixels* en x està desplaçat $avgX$ del centre de la captura del *Wiimote*® (recordem que la resolució és de 1024×768), i convertint aquest desplaçament en un angle mitjançant $radiansPerPixel$, podem aplicar una funció trigonomètrica simple (8.3) per trobar H_x.

$$H_x = \sin(radiansPerPixel * (avgX - 512)) * headDist \tag{8.3}$$

Per a obtenir H_y la idea que seguim és essencialment la mateixa, però amb alguna particularitat que cal tenir en compte.

La ubicació ideal del *Wiimote*® respecte la pantalla és just al centre, però no l'hi podem posar perquè l'interposaríem entre l'usuari i aquesta. Per a evitar-ho, l'haurem de col·locar per sobre o per sota de la pantalla, cosa que introduirà un cert error que caldrà corregir. L'aplicació d'usuari desenvolupada al capítol **??** mostra un cub virtual d'aresta 1. Tenint en compte que hem de situar la càmera virtual al centre de la pantalla, i que el *Wiimote*®, en realitat, està desplaçat mitja aresta amunt o mitja aresta avall de la pantalla, haurem d'aplicar un $factor$ que valgui 0.5 o -0.5 respectivament, per a fer la correcció.

A més a més, degut a la diferència d'altures que hi pot haver entre els usuaris, hem introduït un mecanisme per a establir la posició central de repòs; és a dir, l'usuari se situa a una posició natural davant la pantalla i indica al sistema que aquest és el seu centre, de tal manera que l'aplicació pugui guardar el desplaçament vertical *offset* observat. D'això se'n diu *zeroing y position*, i de l'angle *offset* que trobem en diem $cameraVerticaleAngle$:

$$relativeVerticalAngle = radiansPerPixel * (avgY - 384)$$

$$H_y = factor + \sin(relativeVerticalAngle + cameraVerticaleAngle) * headDist$$

Capítol 9

Part 2: aplicació final

Ja hem introduït anteriorment les parts de xarxa i *head tracking*, que ens facilitaran la comprensió d'aquest capítol. També es recullen en els apèndixs explicacions més concretes sobre els *shaders*, la *WiimoteLib* i el *DirectX*, per ampliar més la informació prèvia a la comprensió d'aquest capítol. Ara es descriurà de forma exhaustiva la segona part del projecte, que consisteix en la implementació d'una aplicació que aprofiti el calibratge fet, tot seguint els passos clàssics del desenvolupament en cascada d'una aplicació.

Primer descriurem quins objectius dels descrits al capítol 1 cal acomplir en aquest punt del projecte. Després es farà un anàlisi més detallat dels requisits funcionals i no funcionals que es té per a aquesta part i que definiran, de forma no ambigua, les característiques de l'aplicació.

Després, quan es tingui ben clar el marc de treball, veurem l'especificació del sistema, el disseny concret que s'ha fet (aplicant el patró de xarxa descrit al capítol 6 i, finalment, comentarem els detalls de la implementació concreta que s'han seguit.

Finalment, parlarem d'alguns dels tests que hem dut a terme per tal d'assegurar-nos de què tot funciona correctament.

9.1 Introducció i objectius

Recordem els objectius descrits en el capítol 1.3, centrant-nos en els que ens interessen per la segona part:

Head tracking Per tal d'assolir un nivell de realisme més elevat, el sistema implementa un mecanisme que fa un seguiment de la posició de l'usuari, de tal manera que el que se l'hi mostra és coherent amb la seva ubicació. En parlem amb més detall al següent capítol i al **??**.

Aplicació d'usuari Necessitem una aplicació que utilitzi tota la infraestructura desenvolupada i que demostri el correcte funcionament del projecte.

El que es vol crear és una aplicació que utilitzi el calibratge fet prèviament i que implementi un sistema de seguiment del cap de l'usuari. Aquesta aplicació haurà d'aprofitar els projectors que han participat a la fase de calibratge per projectar una sola imatge. L'objectiu principal és que l'usuari no sigui conscient que en realitat la pantalla està composta de més d'un projector. Al llarg de la memòria, ja hem anat veient que per fer-ho, necessitem les homografies abans calculades, i per reduir l'efecte del solapament de les àrees de projecció, utilitzarem les màscares cromàtiques.

Per desenvolupar una aplicació que implementés *head tracking* primer calia fer una petita investigació per descobrir quines possibilitats ja existien. A Internet es pot trobar un vídeo d'una aplicació que utilitza el comandament de la consola *Wii* per seguir el cap de l'usuari (l'enllaç del vídeo el podem trobar al capítol B.1), amb el codi disponible. Després d'un estudi d'aquest codi vam concloure que reaprofitar-lo i adaptar la seva solució als nostres objectius era més eficient en temps de desenvolupament que fer-lo enterament nosaltres.

9.2 Especificació

En aquesta part d'especificació, es descriu en profunditat què és el que fa el nostre sistema. Com en el capítol 7.2, s'utilitzen algunes tècniques, com l'anàlisi de requeriments, per definir clara i inambiguament què esperem del nostre sistema.

9.2.1 Restriccions

Començarem per delimitar clarament què és el que farà l'aplicació. D'aquesta manera, comprendrem millor els casos d'ús i ens centrarem en la feina més útil.

Supòsits

Primerament descriurem què estem suposant com a cert en el sistema, i que per tant no hem de comprovar la seva veracitat:

- Disposem de tota la informació obtinguda a partir del calibratge.
- Només els mateixos ordinadors que han participat al calibratge poden executar l'aplicació, amb els mateixos recursos de xarxa.
- Els projectors són els mateixos i estan associats idènticament als ordinadors.
- No s'ha desplaçat cap projector ni en posició ni en orientació, ni ho faran al llarg de l'execució

Aquests supòsits no es deuen tant a que la aplicació no és capaç de funcionar si no es compleixen, sinó més aviat amb el fet que si no es compleix algun d'ells no té sentit continuar, perquè el resultat no serà satisfactori.

Què no fa el sistema

Definit ja què se suposa com a cert abans d'iniciar l'aplicació, passem a puntualitzar què no es pot esperar d'aquesta:

- L'aplicació no calibra el projectors.
- L'aplicació no aplica cap sistema de seguretat en xarxa.

9.2.2 Casos d'ús

Per presentar els casos d'ús de d'aplicació 3D, els dividirem en dues categories: aquells que ja estaven presents a l'aplicació de [CL] i aquells que hi afegim nosaltres per acomplir els nostres objectius.

Al diagrama 9.1 es pot veure en groc els casos d'ús originals i en verd els que hem afegit nosaltres.

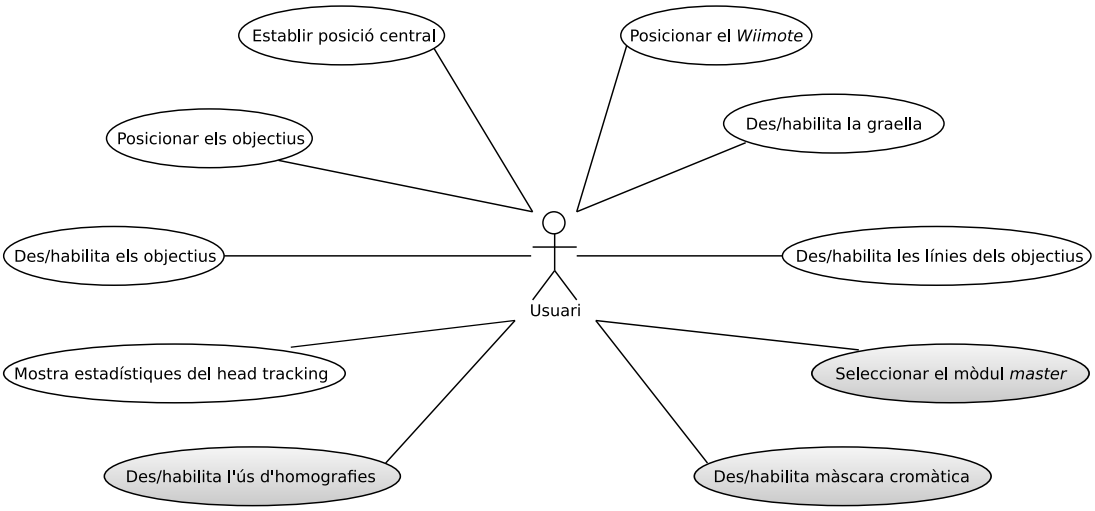


Figura 9.1: Diagrama de casos d'ús de l'*aplicació d'usuari*.

Casos d'ús presents a l'aplicació original

L'aplicació de [CL] ja aportava unes funcionalitats a l'usuari que, descriurem amb més detall a continuació. Aquests casos d'ús que comentem s'han mantingut als nostres canvis degut a la seva utilitat pràctica.

6. Establir posició central	
Descripció	
L'aplicació necessita conèixer quina és la posició inicial des de la qual l'usuari es mirarà la pantalla, la qual podríem considerar equivalent al "centre" d'una *cave*.	
Actor principal	Usuari
Precondició	
El *Wiimote*® veu els dos LED de l'usuari.	
Criteri de validació	
Quan l'usuari se situa a la posició senyalada com "central", l'aplicació mostra l'escena totalment centrada.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari decideix modificar la posició inicial des de la qual mira l'escena. 2. L'usuari se situa a algun lloc de la sala i indica al sistema que aquesta nova posició és la que s'ha de considerar com a inicial.	
	3. El sistema enregistra la nova posició com a posició central. 4. El sistema ubica la càmera virtual al centre, per tal de què la imatge mostrada estigui centrada.
Cursos alternatius	
-	

7. Posicionar Wiimote®	
Descripció	
Com hem avançat al capítol **??**, per tal de calcular bé la posició de l'usuari, la càmera hauria d'estar situada al centre de la pantalla, però això el molestaria i no li deixaria veure les imatges. Per a solucionar-ho, el *Wiimote*® se situa físicament per sobre o per sota de la pantalla, amb la qual cosa caldrà corregir el desplaçament introduït.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
El sistema coneix la ubicació real del *Wiimote*® respecte la pantalla.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari situa el *Wiimote*® per sobre o per sota de la pantalla. 2. L'usuari indica al sistema on l'ha posat.	
	3. El sistema enregistra aquesta posició, de tal manera que podrà aplicar les correccions pertinents.
Cursos alternatius	
-	

8. Posicionar objectius	
Descripció	
Els objectius que mostra l'aplicació poden ubicar-se aleatòriament a diferents posicions del món virtual.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
Els objectius s'han redistribuït de forma aleatòria per l'espai.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari vol canviar la posició dels objectius. 2. L'usuari indica al sistema que vol que els objectius s'ubiquin a noves posicions de l'escenari.	
	3. El sistema redistribueix aleatòriament tots els objectius.
Cursos alternatius	
-	

9. Des/habilita graella	
Descripció	
La sala virtual que es dibuixa està representada amb "filferros". Les parets que la composen no són llises, sinó que poden (o no) mostrar una graella.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
Es mostra o s'oculta la graella de les parets.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari vol amagar (si es veia) o mostrar (si no) la graella de les parets. 2. L'usuari indica al sistema que vol canviar l'estat de la graella.	
	3. El sistema mostra o amaga la graella de les parets.
Cursos alternatius	
-	

10. Des/habilita objectius	
Descripció	
La sala virtual pot o no contenir objectius, i l'usuari hauria de poder seleccionar si els vol veure o no.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
Es mostren o s'oculten els objectius de la sala.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari vol amagar (si es veien) o mostrar (si no) les dianes de la sala. 2. L'usuari indica al sistema que vol canviar l'estat dels objectius.	
	3. El sistema oculta els objectius si es veien, o els mostra (calculant noves posicions) si no.
Cursos alternatius	
-	

11. Des/habilita les línies dels objectius

(continua a la següent pàgina)

(ve de la pàgina anterior)

Descripció	
Tots els objectius poden tenir una línia que va des del seu centre, on estan ubicats "flotant" a l'espai, fins a la paret del fons (uneix el punt central de l'objectiu amb aquest mateix punt projectat al pla de la paret del fons). L'usuari pot fer que aquestes línies es mostrin o s'ocultin.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
Es mostren o s'oculten les línies dels objectius.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari vol amagar (si es veien) o mostrar (si no) les línies de les dianes. 2. L'usuari indica al sistema que vol canviar l'estat de les línies dels objectius.	
	3. El sistema oculta les línies dels objectius si es veien, o les mostra si no.
Cursos alternatius	
-	

12. Mostra/oculta estadístiques del head tracking

Descripció	
L'usuari pot estar interessat en conèixer què tal està funcionant el *head tracking* del sistema, així com algunes estadístiques internes de l'aplicació.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
El sistema mostra algunes estadístiques (nombre de LED que s'estan veient, *frame rate* de l'aplicació, etc.) si no es veien, o les amaga si estaven visibles.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari vol amagar (si es veien) o mostrar (si no) les estadístiques del *head tracking*. 2. L'usuari indica al sistema que vol canviar l'estat de les estadístiques del *head tracking*.	
	3. El sistema oculta les estadístiques si es veien, o les mostra si no.
Cursos alternatius	
-	

Nous casos d'ús

Un dels objectius del projecte era aconseguir una aplicació d'usuari que utilitzés tota la infraestructura desenvolupada; en altres paraules, que "dibuixés" l'escena virtual a la pantalla computada durant els processos de calibratge. A més a més, també volíem que implementés algun mecanisme per a fer el seguiment de l'usuari.

Nosaltres partim d'una aplicació que dibuixa una escena virtual i que, amb el *Wiimote*® fa *head tracking*. El que cal fer, doncs, és afegir-li els components necessaris per a què funcioni de forma "distribuïda", de tal manera que l'escena sigui pintada pels diferents projectors.

Tot seguit, veurem els casos d'ús específics de l'aplicació modificada:

13. Seleccionar el mòdul master	
Descripció	
L'usuari hauria de poder determinar quin mòdul actua com a *master*, de tal manera que la resta siguin de tipus *slave*.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
El mòdul seleccionat actua com a *master*.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari decideix quin mòdul actua com a *master*. 2. L'usuari associa a l'ordinador del mòdul el *Wiimote*® via *Bluetooth*. 3. L'usuari inicia l'aplicació.	
	4. El sistema detecta que té un *Wiimote*® associat. 5. El sistema configura el mòdul per a què treballi com a *master*.
Cursos alternatius	
-	

14. Des/habilita màscara cromàtica	
Descripció	
L'usuari hauria de poder aplicar les correccions cromàtiques que s'han calculat al mòdul de calibratge.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
El mòdul càmera-projector aplica la màscara cromàtica si no s'estava emprant, o deixa d'utilitzar-la altrament.	
Curs típic d'esdeveniments	
Usuari	**Sistema**

(continua a la següent pàgina)

(ve de la pàgina anterior) 1. L'usuari decideix que vol aplicar (o deixar de fer-ho) les correccions cromàtiques en un mòdul determinat. 2. L'usuari indica al mòdul càmera-projector concret que apliqui (o deixi d'aplicar) les correccions cromàtiques.	
	3. El sistema utilitza (o deixa de fer-ho) la màscara cromàtica de què disposa aquell mòdul.
Cursos alternatius	
-	

15. Des/habilita l'ús d'homografies	
Descripció	
L'usuari hauria de poder aplicar les homografies que s'han calculat al mòdul de calibratge.	
Actor principal	Usuari
Precondició	
-	
Criteri de validació	
El mòdul càmera-projector aplica l'homografia si no s'estava emprant, o deixa d'utilitzar-la altrament, de tal manera que, en el primer cas, l'escena 3D es deformi convenientment per adaptar-se a la geometria de la pantalla.	
Curs típic d'esdeveniments	
Usuari	**Sistema**
1. L'usuari decideix que vol aplicar (o deixar de fer-ho) l'homografia en un mòdul determinat. 2. L'usuari indica al mòdul càmera-projector concret que apliqui (o deixi d'aplicar) la seva homografia.	
	3. El sistema utilitza (o deixa de fer-ho) l'homografia de què disposa aquell mòdul.
Cursos alternatius	
-	

9.2.3 Anàlisi de requisits

Com ja hem vist en capítols anteriors, l'SRS (*Software Requirements Specification*, anàlisi de requisits) és una eina que permet posar de manifest d'una forma clara i concisa aquelles funcionalitats que el sistema ha d'oferir, així com aclarir aquelles restriccions a les que estigui subjecte.

El resultat d'aquest anàlisi no només emmarca perfectament l'àmbit de treball, sinó que també es converteix en una eina que farà possible comprovar de forma inequívoca si el projecte fa tot el que ha de fer.

És important comentar que l'SRS conté només requisits funcionals i no funcionals, però no esbossa cap tipus de solució subjecta a la tecnologia, essent totalment independent de la solució final proposada.

Per tal de fer l'anàlisi de requisits s'ha seguit la plantilla del mètode *Volere*[], adaptant-la convenientment als nostres interessos, atès que es tracta d'un mètode molt exhaustiu i que, en general, genera una documentació molt extensa.

9.2.4 Requisits funcionals

En aquest apartat es descriuen els requisits funcionals que es deriven dels casos d'ús que hem afegit a l'aplicació original (és a dir, els descrits a l'apartat 9.2.2), així com els que detectem a partir de la pròpia definició dels objectius.

Hem decidit no posar els requisits que s'obtindrien dels casos d'ús originals, perquè compliquen innecessàriament la memòria i suposen una feina addicional que no ens ajudarà a l'hora de aplicar els canvis que siguin pertinents:

Requisit 1			
Tipus:	Funcional	**Casos d'ús associats:**	∼
Descripció			
El *master* ha de conèixer les IP de tots els mòduls *slave*.			
Justificació			
Per tal de passar-los-hi la ubicació de l'usuari (del qual n'està fent el seguiment amb el *Wiimote*®), necessita conèixer la seva direcció a la xarxa.			
Condició de satisfacció			
-			
Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	-	**Conflictes:**	-

Requisit 2			
Tipus:	Funcional	**Casos d'ús associats:**	13
Descripció			
L'usuari hauria de poder seleccionar quin mòdul actuarà com a *master*.			
Justificació			
Un dels mòduls ha de ser qui tingui associat el *Wiimote*® i actui com a *master*.			
Condició de satisfacció			
-			
Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	-	**Conflictes:**	-

Requisit 3			
Tipus:	Funcional	**Casos d'ús associats:**	13

(continua a la següent pàgina)

(ve de la pàgina anterior)

Descripció

Els mòduls que no actuin com a *master* s'han de configurar com a *slave* automàticament.

Justificació

Els mòduls *slave* no tenen un *Wiimote*® associat i, per tant, algú els hi ha de proporcionar la informació de la ubicació de l'usuari.

Condició de satisfacció

-

Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	-	**Conflictes:**	-

Requisit 4

Tipus:	Funcional	**Casos d'ús associats:**	∼

Descripció

El sistema hauria de poder carregar la informació que ha calculat el mòdul de calibratge.

Justificació

-

Condició de satisfacció

-

Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	**??**	**Conflictes:**	-

Requisit 5

Tipus:	Funcional	**Casos d'ús associats:**	14

Descripció

L'usuari hauria de poder aplicar una màscara cromàtica en un mòdul determinat.

Justificació

-

Condició de satisfacció

La imatge projectada pel mòdul concret té aplicada una màscara a sobre.

Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	-	**Conflictes:**	-

Requisit 6

Tipus:	Funcional	**Casos d'ús associats:**	14

Descripció

L'usuari hauria de poder desactivar l'ús d'una màscara cromàtica en un mòdul determinat.

Justificació

-

(continua a la següent pàgina)

(ve de la pàgina anterior)

Condició de satisfacció			
La imatge projectada per un dels mòduls no té aplicada cap correcció cromàtica.			
Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	-	**Conflictes:**	-

Requisit 7			
Tipus:	Funcional	**Casos d'ús associats:**	15
Descripció			
L'usuari hauria de poder aplicar en un mòdul determinat la seva homografia.			
Justificació			
-			
Condició de satisfacció			
La imatge projectada pel mòdul es deforma convenientment, segons l'homografia.			
Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	-	**Conflictes:**	-

Requisit 8			
Tipus:	Funcional	**Casos d'ús associats:**	15
Descripció			
L'usuari hauria de poder desactivar l'ús de l'homografia en un mòdul determinat.			
Justificació			
-			
Condició de satisfacció			
La imatge projectada per un dels mòduls mostra tota l'escena virtual, sense aplicar-se-li cap homografia.			
Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	-	**Conflictes:**	-

9.2.5 Requisits no funcionals

Finalment, es veuran els requisits no funcionals que caracteritzen l'aplicació d'usuari.

De fet, tenint en compte que aquesta part del projecte només pretén demostrar la viabilitat de tot plegat, no disposa d'uns requisits no funcionals clars. No obstant, hem decidit que els canvis que apliquéssim al programa original tinguessin algunes propietats que els fessin útils en un futur:

Requisit 9

(continua a la següent pàgina)

(ve de la pàgina anterior)

Tipus:	Plataforma	**Casos d'ús associats:**	-
Descripció			
El sistema ha de funcionar en un entorn distribuït.			
Justificació			
Els mòduls de què disposem són, tots ells, ens independents. Per tant, i ja de partida, ens trobem en un entorn distribuït amb el qual el nostre sistema s'ha d'enfrontar.			
Condició de satisfacció			
-			
Satisfacció del client:	5	**Insatisfacció del client:**	5
Dependències:	-	**Conflictes:**	-

Requisit 10			
Tipus:	Plataforma	**Casos d'ús associats:**	-
Descripció			
El sistema ha de poder estendre's fàcilment.			
Justificació			
Tot i que l'aplicació d'usuari només és una *demo*, és interessant que per a desenvolupar-ne una de nova es pugui aprofitar el màxim de codi possible, sobretot el pertanyen a la xarxa o a l'aplicació d'homografies i màscares.			
Condició de satisfacció			
-			
Satisfacció del client:	5	**Insatisfacció del client:**	1
Dependències:	-	**Conflictes:**	-

9.2.6 Diagrama de classes

En aquest apartat es mostra el diagrama de classes que dóna suport a tot el sistema. Es veuran els diferents conceptes que el composen i les relacions entre ells.

Una de les eines que tenim per a representar-ho és l'UML (*Unified Modelling Language*, Llenguatge de modelatge unificat), el qual ens permet expressar aquests conceptes i relacions de forma visual .

Per tal de simplificar la comprensió del diagrama, es dividirà en petites unitats, que mostrem en colors diferents, i les descriurem després una a una, comentant els punts més importants.

Aquest diagrama intenta respectar al màxim els requisits no funcionals de reusabilitat i ampliabilitat. Tal i com està plantejat aquest diagrama, per fer una aplicació totalment diferent, però que utilitzés el *head tracking* i el nostre sistema de xarxa, només li caldria dissenyar i implementar les subclasses. Es veu també que l'aplicació 3D es composa de vàries parts, i al llarg d'aquesta secció anirem descrivint-les una a una. Identifiquem-les:

- Interfície
- Entrada/Sortida
- Xarxa

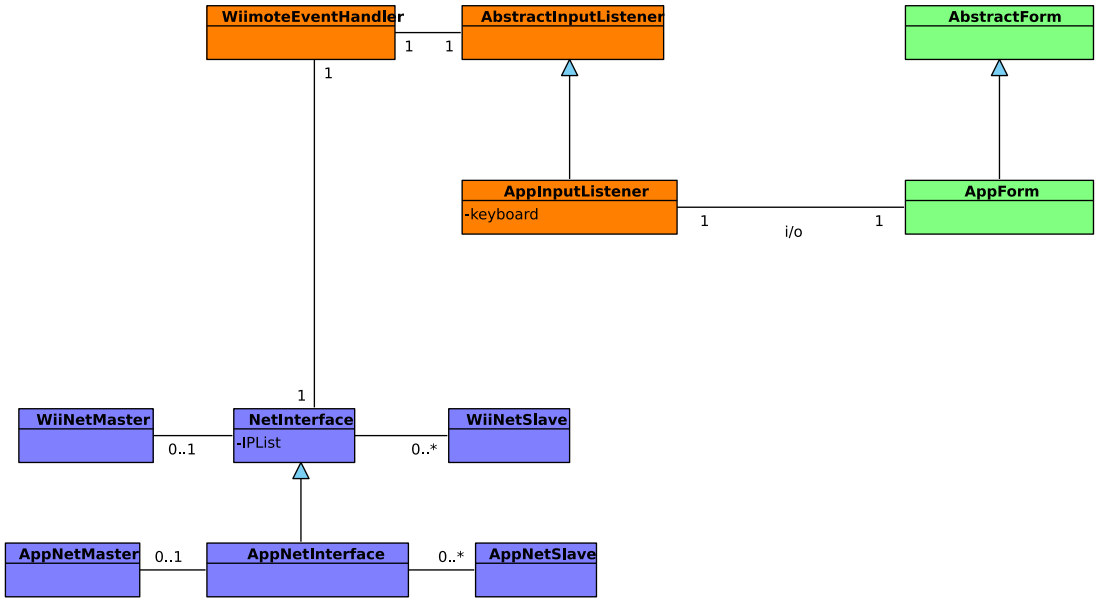


Figura 9.2: Diagrama de classes d'especificació.

Per fer més fàcil al lector entendre i identificar les parts per separat, s'inclouen les associacions amb alguns elements externs a la part en concret que estem explicant, a part de mantenir els colors del diagrama complert.

Interfície

Aquest és un apartat crític, perquè tenim com a requisit no funcional que sigui ampliable a futures aplicacions que utilitzin el *head tracking*, i com a requisit funcional que tot funcioni en temps real.

Per complir amb el requisit no funcional, crearem una estructura d'herències que reculli la informació del *head tracking* independentment de què valguem fer amb la informació (vegeu la figura 9.3). A aquesta subclasse li diem de moment *AppForm*, i més endavant, quan tinguem més detalls de què volem mostrar-li a l'usuari, la reanomenarem. Així doncs, aquest *AbstractForm* és qui pren la informació *head tracking* per tal *AppForm* no se n'hagi de preocupar.

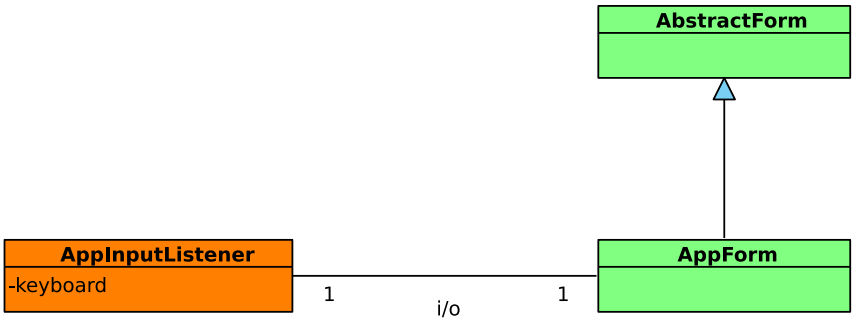


Figura 9.3: Diagrama de classes d'especificació del mòdul de interfície.

Homografies i Màscares

Les homografies i les màscares són parts fonamentals per què l'usuari pugui veure bé la pantalla. Hem d'utilitzar-les de manera molt relacionada amb com fem la interfície, per tant, a nivell d'especificació només veurem com es relacionen (vegeu figura 9.4). Dins de *Calibration* trobem les homografies i les màscares, i és *AbstractForm* qui s'encarrega de fer-les servir, per aconseguir que el desenvolupador no hagi de preocupar-se per aquest aspecte.

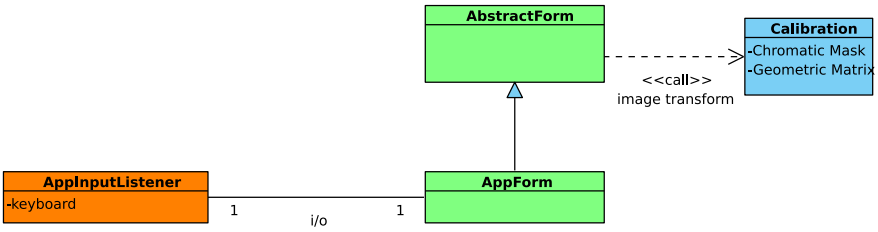


Figura 9.4: Diagrama de classes d'especificació del mòdul de interfície amb homografies i màscares.

Entrada / Sortida

A través dels casos d'ús i els requisits ja ens hem assabentat de quines entrades i sortides necessitem. Bàsicament les entrades ens defineixen les homografies i les màscares cromàtiques, la descripció de la pantalla i altres dades necessàries pel *head tracking*, així com les peticions de l'usuari a través de la interfície i les dades que enviem i rebem per xarxa.

A la figura 9.5 podem veure el lector de dades, que n'hem dit *ConfigLoader*, i les diferents interfícies per les quals enviem i rebem les dades, com són el teclat que tenim a *AppInputListener*, o també les dades del *Wiimote*, que tenim a *WiimoteEventHandler*.

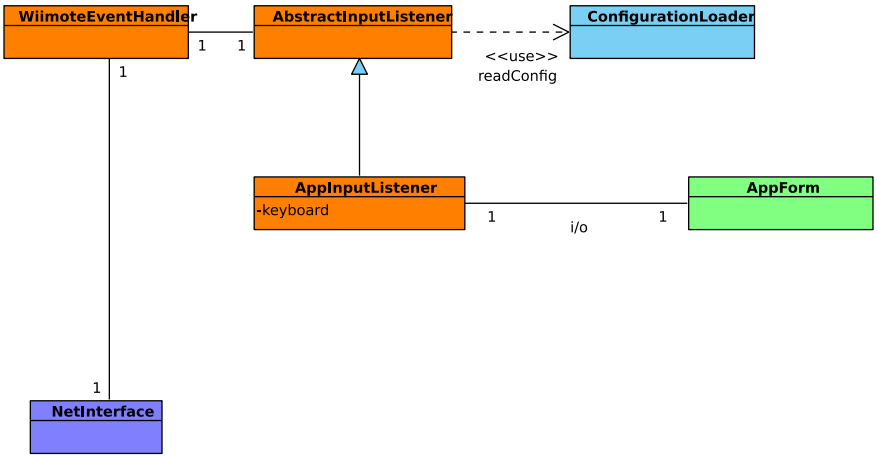


Figura 9.5: Detall del diagrama de classes d'especificació: entrada i sortida de dades

Xarxa

La idea de la xarxa està en la mateixa línia que les altres. També es vol que el sistema sigui ampliable per la xarxa, perquè una altra aplicació pot voler utilitzar la xarxa per enviar dades diferents, i per això proporcionem la infraestructura per tal que el desenvolupador només s'hagi de preocupar per saber quines dades enviar, i què fer amb elles després (vegeu la figura 9.6). *NetInterface* s'ocupa de sincronitzar tots els equips perquè tots funcionin alhora. Si l'aplicació necessita la xarxa per passar més informació, pot heretar de *NetInterface* a *AppNetInterface* i utilitzar els seus recursos, o implementar diferents tipus de xarxa, que aquí en diem *AppNetMaster* i *AppNetSlave*.

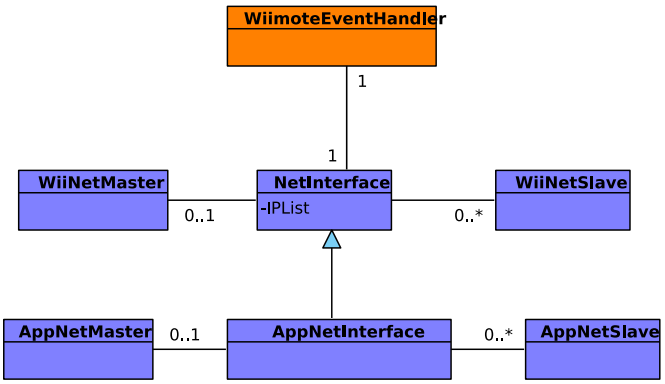


Figura 9.6: Detall del diagrama de classes d'especificació: xarxa

9.3 Disseny

En aquesta secció es parla de quin disseny s'ha proposat per l'aplicació. Veurem que el fet que bona part del codi ja estigués implementat ha condicionat fortament les decisions que hem pres. Aquest context de desenvolupament propícia que el disseny sigui molt atípic, ja que la fase posterior, la implementació, està feta amb anterioritat.

9.3.1 Estudi del codi existent

A la primera ullada al codi, ja es va veure que hauríem d'introduir-hi molts canvis. L'aplicació de [CL] funcionava correctament i tenia quasi tots els elements que es necessitaven, però tenia tot al mateix lloc, sense cap tipus d'organització. Com partíem d'una sola classe, la nostra primera feina de disseny era veure quins elements de l'especificació ja estaven implementats, i, a partir d'aquesta informació, fer un primer diagrama de disseny (vegeu figura 9.7).

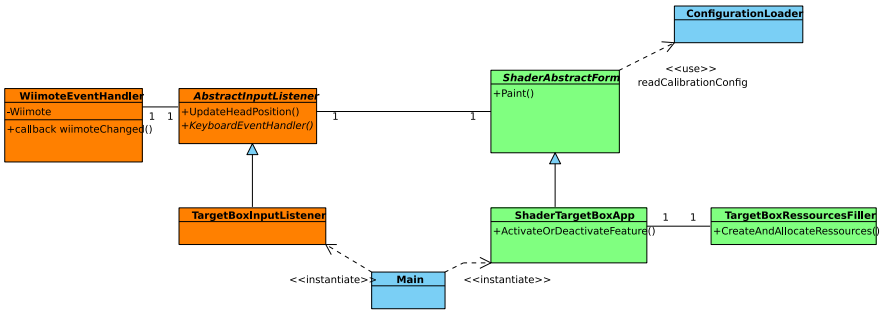


Figura 9.7: Primer diagrama de disseny.

Aprofitem per comentar què és el que es mostra, finalment, en aquesta aplicació. Es tracta d'un cub, d'aresta 1, on dins hi ha unes dianes en posicions aleatòries, que estan unides per rectes a una de les parets, que anomenarem fons. L'usuari veu el cub des de la paret oposada, que és transparent. D'aquesta manera, es té una sensació d'estar mirant una habitació a través de la pantalla. Aquesta sensació es multiplica quan activem el *head tracking*, i parlem ja d'una bona immersió. Les nostres ampliacions no modificaran res de l'anterior, de fet busquem que l'usuari tingui la mateixa sensació, però amb una pantalla composada per N projectors.

9.3.2 Ampliacions del codi existent

El diagrama de la figura 9.7 no és suficient per complir tots els requisits que tenim. Falten precisament les parts que li mancaven a l'aplicació de [CL], en concret la xarxa i la utilització del calibratge. També s'ha de respectar els requisits no funcionals, i en concret a l'hora de pensar el disseny es tindrà present el requisit que l'aplicació sigui canviable i ampliable. El diagrama de la figura 9.12 que s'ha dissenyat pretén respectar tot l'esmentat.

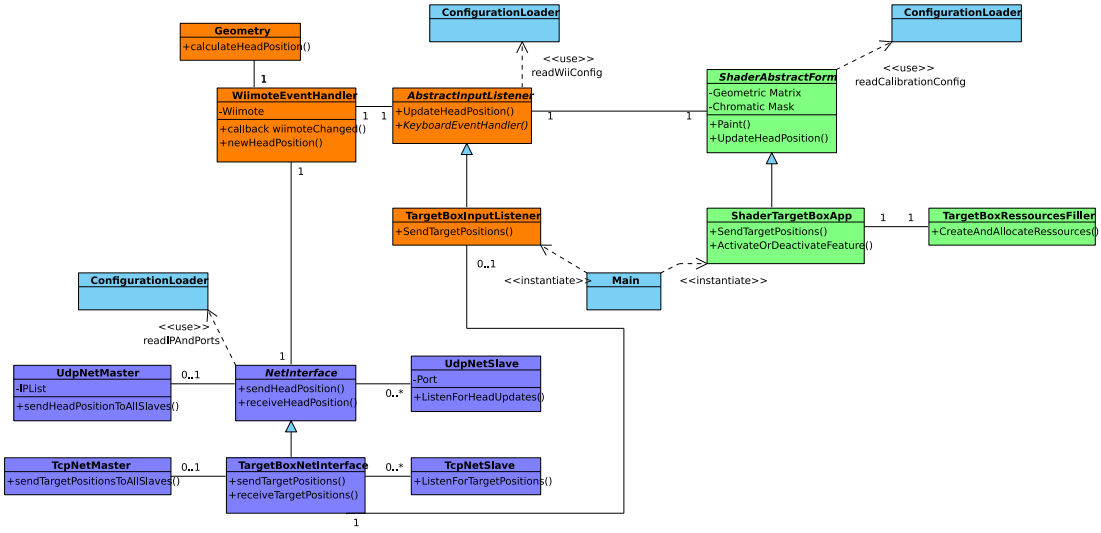


Figura 9.8: Diagrama de disseny complet.

9.3.3 Arquitectura del sistema

L'arquitectura en tres capes (presentació, domini, gestió de dades) permet una separació molt clara en tot moment. És l'arquitectura més comú i potser també la més intuïtiva. El problema és que en aquest cas no es necessita una gestió de dades gaire complexa, però, en canvi, sí cal una xarxa, i volem que mantingui una independència amb la resta de l'aplicació. D'aquesta manera, substituint la capa de gestió de dades per una capa de xarxa, mantenim una aplicació canviable i ampliable. En el diagrama 9.12 s'han separat les capes en tres colors diferents: el verd per la capa de presentació, el taronja per la de domini, i la blava fosca per la de xarxa. El color blau cel l'hem reservat per elements auxiliars que no formen una capa concreta.

De la mateixa manera que ho hem fet a l'especificació, per fer més fàcil al lector entendre i identificar les parts per separat, s'inclouen les associacions amb elements externs a la capa en concret que s'estiguin explicant, a part de mantenir els colors del diagrama complert.

Capa de presentació

La capa de presentació s'encarrega de mostrar una interfície a l'usuari, de manera que pugui interactuar amb l'aplicació i veure els resultats. Pel que fa al disseny, descrivim quina serà l'estructura mitjançant la qual l'usuari pugui fer tots els casos d'ús definits anteriorment. Cal esmentar que com les dades del *Wiimote* es reben a través d'una llibreria externa, aquesta part no la inclourem dins d'aquesta capa.

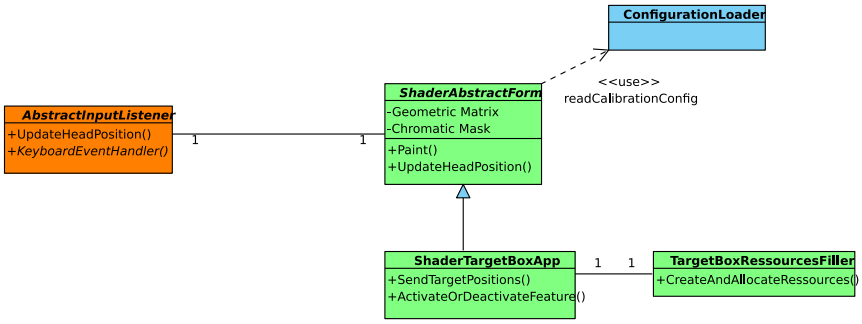


Figura 9.9: Diagrama de disseny de la capa de presentació.

La classe `AbstractForm` crida a `ConfigurationLoader` per recollir les dades del calibratge, i recull la posició del cap de l'usuari necessària per situar la càmera virtual. `ShaderTargetBoxApp` és qui defineix què es mostra realment i quina aplicació s'està fent realment. Aquí és on es posa el mon virtual que havia creat [CL]. Hem decidit separar la creació dels elements en una classe apart, `TargetBoxRessourcesFiller`, en previsió de que `ShaderTargetBoxApp` pot ser una classe bastant complexa, vist el codi de [CL].

Capa de domini

La capa de domini s'encarrega de gestionar els càlculs, crear les estructures necessàries, instanciar tot el necessari, etc. En el nostre disseny, s'encarrega sobretot de gestionar les comunicacions entre les capes de presentació i de xarxa.

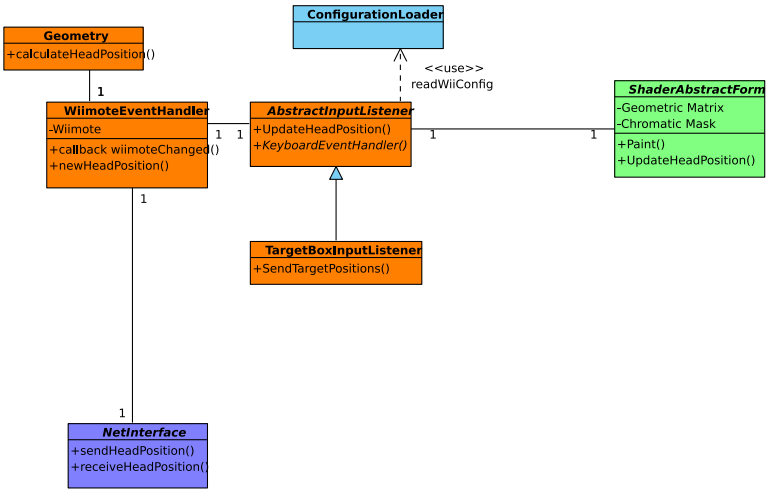


Figura 9.10: Diagrama de disseny de la capa de domini.

La feina de càlcul recau sobretot a `Geometry`, que rep des dades de la posició dels LED i ho transforma en coordenades 3D. L'encarregat de rebre les dades del *Wiimote* és `WiimoteEventHandler` i qui recull els events de teclat és `TargetBoxInputListener`, que implementa la funció de la superclasse. Com hem comentat en el capítol de *head tracking*, necessitem algunes dades, que llegirem amb la classe `ConfigurationLoader`

9.3.4 Capa de gestió de xarxa

La capa de gestió de xarxa està dissenyada per complir amb el requisit no funcional de ser ampliable i reutilitzable. Utilitzem el patró controlador-façana, que veurem més endavant.

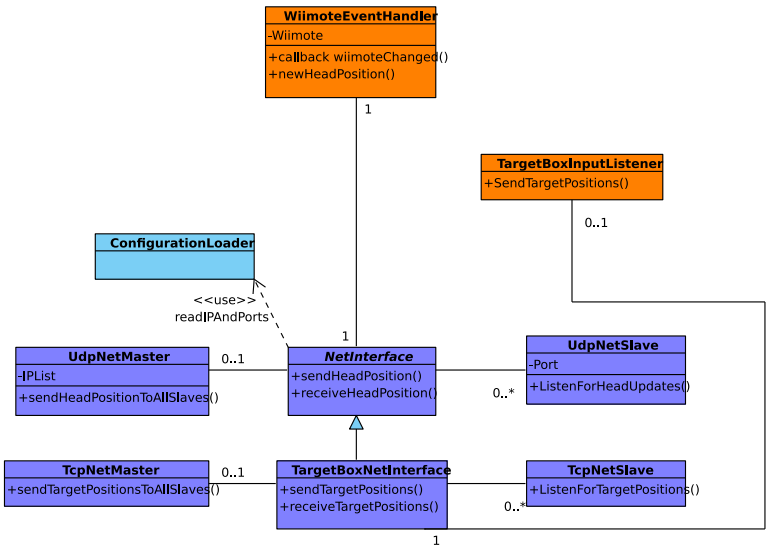


Figura 9.11: Diagrama de disseny de la capa de xarxa.

9.3.5 Patrons aplicats

La raó per la qual s'utilitzen patrons en aquest disseny és per complir amb els requisits no funcionals. Expliquem a continuació amb més detall els patrons que fem servir al disseny. Abans, però volem aclarir que la classe ConfigurationLoader que veiem repetida al diagrama 9.12, no és cap patró. Podríem pensar en aplicar un patró *Singleton*, però no seria una bona idea. La raó és que ocuparia una memòria innecessària, ja que aquesta classe llegeix uns fitxers de configuració al arrancar l'aplicació i no es torna a fer servir mai més en aquella execució.

Controlador-Façana

El patró controlador façana és molt útil per mantenir independència entre diferents parts d'un sistema. És molt habitual trobar-lo en arquitectures de tres capes, com la nostra. En el nostre cas, aquest patró l'hem utilitzat per la xarxa, aconseguint així una bona independència amb la resta de l'aplicació, i no necessitar saber qui és el màster, per exemple. Per més informació vegeu el capítol 6.2.1. El que podem veure al diagrama 9.11 és que `NetInterface` és la nostra façana. La resta de l'aplicació no ha de preocupar-se d'on venen les dades, perquè sempre passen per la façana.

Patró *màster-slave*

Aquest patró és més un disseny de la xarxa que no pas un patró de disseny. Tenim com a requisits i precondicions que un i només un ordinador tindrà un *Wiimote*, i aquest haurà de transmetre la informació a la resta. A nivell de disseny, només hem d'especificar que un màster pot tenir molts esclaus. Per més informació vegeu el capítol 6.2.1. Per intentar aclarir-ho però

vegem la figura 9.12. En aquest diagrama, veiem l'execució de l'ordinador 1, que ha detectat que té un *Wiimote* connectat. Aquest i només aquest envia les dades que la resta d'ordinadors necessiten, i la resta escolten.

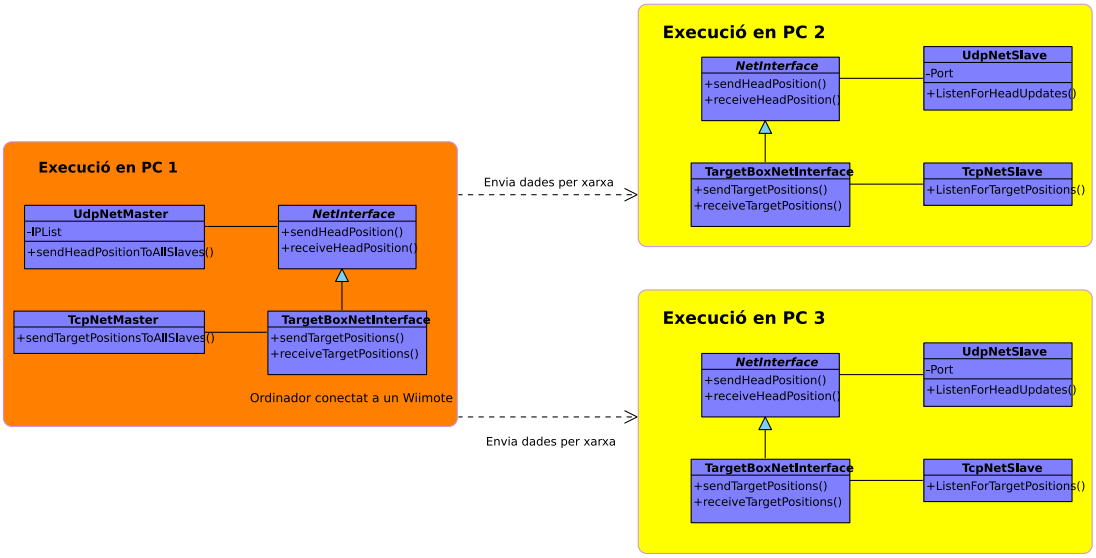


Figura 9.12: Esquema de xarxa amb el patró màster-esclau.

9.4 Implementació

Ara es parlarà de com ha estat el procés d'implementació, i comentarem alguns detalls concrets que creiem útils remarcar, ja sigui per entendre millor tota l'estructura de l'aplicació com per veure els efectes concrets d'una part sobre el resultat final.

9.4.1 Punt de partida

El nostre punt de partida ha estat l'aplicació de [CL]. Aquesta aprofita el comandament de la consola *Wii* per capturar la posició del cap respecte la pantalla i així implementar un sistema de *head tracking*. El codi és lliure està disponible gratuïtament a la pàgina de l'autor.

Aquest codi està desenvolupat en C#, i al ser un llenguatge amb una sintaxi molt pròxima al Java i tenir molta feina ja feta, es va decidir adaptar aquell codi als nostres requeriments i no refer-ho tot amb un altre llenguatge.

9.4.2 Utilització del *Wiimote*

Per tal de poder utilitzar el comandament de la consola *Wii*, hem interactuat amb la llibreria *WiimoteLib* (vegeu l'apèndix B). La nostra feina ha estat programar unes funcions que recollissin les dades de la càmera d'infrarojos i manipular-les fins a obtenir unes coordenades en tres dimensions. Aquestes funcions actuen com a *callbacks*, que s'executen només quan rebem un event determinat. Un exemple de la manera de dir-li a la *WiimoteLib* quins són aquests *callbacks* és aquest: `remote.OnWiimoteChanged += new WiimoteChangedEventHandler (wm_OnWiimoteChanged);` , on remote és la variable que representa la connexió amb el

Wiimote, `OnWiimoteChanged` és el tipus d'event i `wm_OnWiimoteChanged` és la funció *callback* que programa què farem quan arribi l'event `OnWiimoteChanged`.

9.4.3 Implementació del patró *master - slave* en C#

La implementació del patró *master - slave* en C# s'ha integrat dins d'un altre patró: el façana. D'aquesta manera, la resta de l'aplicació no ha de saber si estem treballant com a màster, com esclau, i ni tan sols amb quines màquines estem connectats, si és que ho estem. Pel que fa a la xarxa, s'han utilitzat sockets tcp i udp. El màster declara un *array* de *sockets*, i intenta connectar-se a tots ells. Com les peticions de connexió són bloquejants fins que algú l'accepti, ho fem en *threads* independents. Per més informació, vegeu el capítol 6.2.1

Real time master - slave

És un cas particular del patró *master - slave* és que s'utilitza per dades en temps real, i no és greu perdre alguna dada. El més important és tenir totes les execucions en tots els ordinadors el més sincronitzades possible.

Per més informació, vegeu el capítol 6.3.2

9.4.4 DirectX i *shaders*

El codi del qual es va partir utilitzava DirectX per mostrar la seva interfície. La nostra tasca era deformar la imatge del *frame buffer* per tal que l'usuari la veiés correctament. Com sempre, hi havia vàries alternatives per complir aquesta missió, però aquest cop la decisió va ser ben fàcil: utilitzaríem *shaders*. La raó és que si es genera una imatge i se li aplica un filtre de *warping* (deformació de imatge), cada *frame* que enviem consumirà uns recursos i un temps totalment inadmissibles, cosa que pràcticament descartaria fer una aplicació en temps real. Els *shaders* s'executen a la targeta gràfica i consumeixen molts menys recursos. Per exemple, les correccions geomètriques només s'apliquen en els vèrtexs, i les correccions de color només en les superfícies. Per més informació sobre com s'apliquen aquestes matrius vegeu l'apèndix D

Homografies

Les homografies que s'han calculat en el pas previ són absolutament necessàries per saber com deformar la imatge de sortida per tal que l'usuari la vegi correctament. En el nostre cas, però, només necessitem saber la matriu, i cap més informació associada. S'utilitza la classe `Matrix` de *DirectX* per operar amb les homografies. Creiem necessari esmentar-ho en particular perquè són una part molt important del sistema.

Màscares

Les màscares són l'entitat que ens permet dur a terme la correcció cromàtica. El que fan és modificar el color de la imatge que s'envia a projectar, però, un cop més, s'utilitzarà una estructura de dades ja feta per emmagatzemar aquestes dades i operar amb elles. Aquesta estructura es diu `Texture`, i la raó de perquè la fem servir és que la podem passar directament a la targeta gràfica d'una manera molt senzilla. Per aplicar la màscara dibuixem un rectangle que ocupa tota la pantalla i que se situa sempre davant de la imatge. En aquest rectangle hi apliquem la textura creada a partir de la màscara obtinguda. Com la màscara està en escala de grisos, la idea és pintar-lo de negre i utilitzar aquests grisos com transparències en el canal alpha, o *alpha blending*. És a dir, a la part on hi ha solapament, es desitja que la màscara vagi disminuint la intensitat lumínica del projector. Els valors d'alpha van augmentant i per tant la

textura és progressivament més opaca. Al ser de color negre, quan més opaca és, menys llum s'acaba emetent.

9.5 Proves

En aquesta secció detallem quines proves hem fet al llarg de tot el procés per tal de determinar que tots els components de l'aplicació són estables i robusts als canvis i interaccions amb l'usuari.

9.5.1 Proves unitàries

Com hem partit d'un sistema que ja funcionava, el procés de desenvolupament ha estat relativament més fàcil. La feina s'ha basat principalment en afegir funcionalitats a aquest programari, i a cada pas hem anat provant que tot funcionés correctament.

Lectura de dades

La primera tasca que vam desenvolupar va ser separar la part de lectura de dades de la resta del codi. Vam crear la classe `ConfigLoader` i vam provar-la de diferents maneres. Les proves han estat sobretot per corregir la resistència a errors en els fitxers. De totes maneres, aquest no és un punt clau, perquè els fitxers que nosaltres llegim ara, els generem a l'etapa de calibratge, que ja hem depurat prèviament.

Xarxa

Una de les parts que tradicionalment és més susceptible a errades és, precisament, la xarxa. Qualsevol sistema que incorpori comunicació entre diferents ordinadors requereix un exhaustiu control d'errors i precaució per la seguretat. Ara bé, ja hem dit que el nostre sistema no contemplaria un control de seguretat per caure fora dels objectius del projecte. Per tant, la feina de control i proves es redueix bastant.

El que hem provat ha estat primerament que sempre hi hagués connectivitat, independentment de qui es connectés primer, si el màster o els esclaus. També hem provat que els intents de connexió no fossin bloquejants, és a dir que l'aplicació pogués seguir funcionant encara que no estigués connectada amb ningú.

També hem fet proves per la xarxa amb temps real. En general, suposem que les condicions de la xarxa física no seran cambiants dràsticament, i que per tant no hi hagi canvis molt grans amb les latències de xarxa. Segons les proves, tres segons és una xifra acceptable per recalcular aquesta latència i sincronitzar així els equips. D'aquesta manera, no hi ha un *overhead* gran per mantenir aquest càlcul al dia, i errors momentanis a la xarxa no triguen en solucionar-se.

Integració amb el *Wiimote* i la *WiimoteLib*

Aquesta part era la que a priori semblava més incerta, sobretot pel nostre desconeixement, ja que era el primer cop que treballàvem amb la *WiimoteLib*, i no sabíem quina estabilitat ni fiabilitat tenia. La nostra agradable sorpresa ha estat comprovar que és una llibreria molt ben feta i molt fàcil d'utilitzar, i que per tant les proves que hem hagut de fer per comprovar el bon funcionament han estat mínimes. A més a més, bona part del codi d'aquesta part ja estava fet, i només l'hem hagut de reutilitzar.

9.5.2 Proves d'integració

Havent fet proves unitàries per cada part del projecte, les proves d'integració es redueixen considerablement. L'objectiu és comprovar que totes les parts es comuniquen bé i que fan la seva tasca igual de bé que a les proves unitàries.

El nostre mètode per comprovar la integració ha estat molt senzill: hem anat comprovant sempre que cada part que afegíem, no només funcionava bé per sí sola, sinó que s'acoplava bé a la resta. El fet que cada part sigui suficientment independent per la resta com per al menys fer les seves proves, ha possibilitat aquest mètode.

Aquest mètode s'ha demostrat efectiu pel nostre cas, perquè provar cada part per separat no ha implicat un volum de feina important, i provar com s'integrava cada part amb la resta tampoc ha suposat un increment excessiu de temps. A canvi, un cop acabades les proves de l'última part, ja vam poder concloure que el sistema funciona bé i no calia dedicar-hi temps a corroborar i comprovar la integració total.

Capítol 10

Prototipus

En aquest capítol es descriu breument com s'han construït cadascun dels mòduls que conformen el prototipus físic del projecte.

10.1 Introducció

Fins ara, hem vist que cadascun dels prototipus consta de les següents peces:

- Un ordinador portàtil.
- Un projector
- Una càmera.

Tenint en compte que el procés de calibratge ha de ser força acurat, interessa que tots els mòduls estiguin muntats sobre alguna plataforma que eviti que es puguin moure lliurement, i que doni certa llibertat a l'hora d'ubicar-los per la sala. A més a més, el projector i, eventualment, el portàtil, necessitaran connectar-se al corrent elèctric, i no es pot assumir que qualsevol sala disposarà de suficients endolls.

Tenint en compte aquestes consideracions, es decideix construir una plataforma que, si bé senzilla, disposi de, com a mínim:

- Un suport on recolzar el projector.
- Un suport on recolzar el portàtil.
- Un lladre que permeti endollar el projector, el portàtil i, eventualment, el lladre d'un altre mòdul.
- Un suport mòbil i extensible per a la càmera.

10.2 Construcció

La figura 10.1 mostra un dels prototipus construïts. Aquest consta d'una base on es recolza el projector feta de metacrilat, el qual és prou resistent com per a desplaçar-lo per la sala sense que es trenqui. Podem veure com la base suporta el lladre que ens permet realitzar les connexions, així com un "braç" d'alumini on s'ha collat la càmera. Sobre d'aquesta, en un segon nivell,

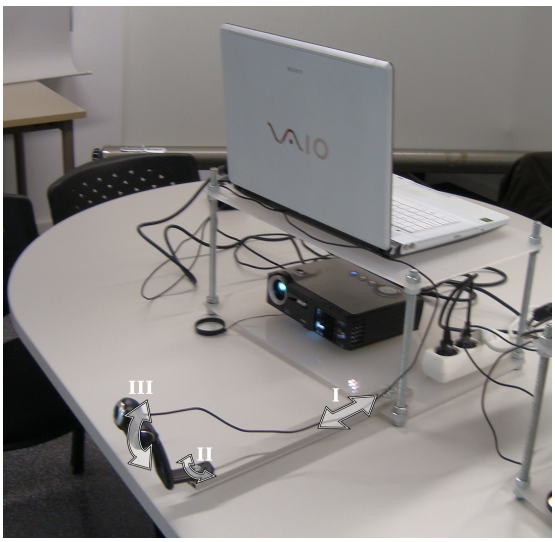


Figura 10.1: Fotografia d'un dels prototipus muntats.

s'ubica una segona planxa de metacrilat on es pot recolzar el portàtil. Ambdues planxes, unides mitjançant les barres metàl·liques, esdevenen un mòdul ben sòlid i segur.

El braç que suporta la càmera ofereix diversos graus de llibertat, de tal manera que la podem apropar i allunyar respecte el focus del projector (I), rotar-la lleugerament (II) o fer que enfoqui més amunt o més avall (III).

Capítol 11

Cost del projecte

En aquest capítol s'ha dut a terme un breu anàlisi de costos del projecte, on es calcula, per una banda, el cost dels treballadors que hi haurien estat implicats i, per l'altra, el del maquinari que s'ha hagut d'adquirir.

És important adonar-se de la importància que té haver realitzat la planificació correctament per tal de calcular amb precisió el cost del projecte abans de començar-lo. Haver fet una bona planificació al principi de les tasques i les seves duracions, així com preveure qui se n'hauria d'encarregar, permet estimar molt bé el cost que caldria assumir per a desenvolupar-lo.

En aquest cas, però, s'ha utilitzat la seva duració real[1], enlloc de l'estimació inicial. D'aquesta manera, el que obtenim al final és el cost real que representa.

11.1 Cost salarial

Primerament, cal notar que les tasques necessàries per a dur a terme el projecte requereixen l'acció de treballadors amb diferents habilitats. En el nostre cas, aquestes quedaven cobertes amb dos rols[2]: un *analista* i un *programador*.

11.1.1 Dedicació dels treballadors

A cadascun dels rols se l'hi assigna les tasques que li pertoca desenvolupar i, per cadascuna, en quina mesura ho fa. Això permet determinar quantes hores ha dedicat a una tasca en concret i, conseqüentment, quantes hores ha dedicat en total al projecte. Tota aquesta informació la podeu veure recopilada a les taules 11.1 i 11.3.

Així, per exemple, si algú ha dedicat dos dies de feina a una tasca determinada, és a dir, un total de setze hores, però només amb un 50% de dedicació, hi hauria dedicat un total de $16h \times 50\% = 8h$.

Activitat	**Dies**	**Dedicació**	**Hores**
Especificació del mòdul de calibratge	67	40%	215
Disseny del mòdul de calibratge	65	40%	205

(continua a la següent pàgina)

[1] Vegeu la secció 4.5.

[2] De fet, és possible considerar un tercer rol, encarregat de la construcció dels prototipus. Aquest, però, s'ha ignorat a l'hora de realitzar els càlculs ja que, tal i com es pot veure més endavant, s'assumeix que aquesta tasca la fa el programador.

(ve de la pàgina anterior)

Especificació de l'aplicació d'usuari	45	30%	110
Disseny de l'aplicació d'usuari	8	40%	25
		Total d'hores	555

Taula 11.1: Hores invertides per l'analista al projecte.

Activitat	**Dies**	**Dedicació**	**Hores**
Desenvolupament calibratge geomètric	35	60%	170
Desenvolupament calibratge cromàtic	25	60%	120
Correccions necessàries al calibratge	30	60%	140
Escriptura de resultats	3	40%	10
Desenvolupament del mòdul de visualització	67	60%	320
		Total d'hores	760

Taula 11.2: Hores invertides pel programador al projecte.

11.1.2 Salaris

Després de conèixer la dedicació en hores al projecte de cada treballador, cal saber quina és la seva retribució per hores per tal de poder calcular-ne el cost.

Per tal de fer-nos una idea del salari mig d'un analista i d'un programador s'ha consultat [IJT]. A la següent figura es pot veure l'evolució dels salaris de cadascun d'ells:

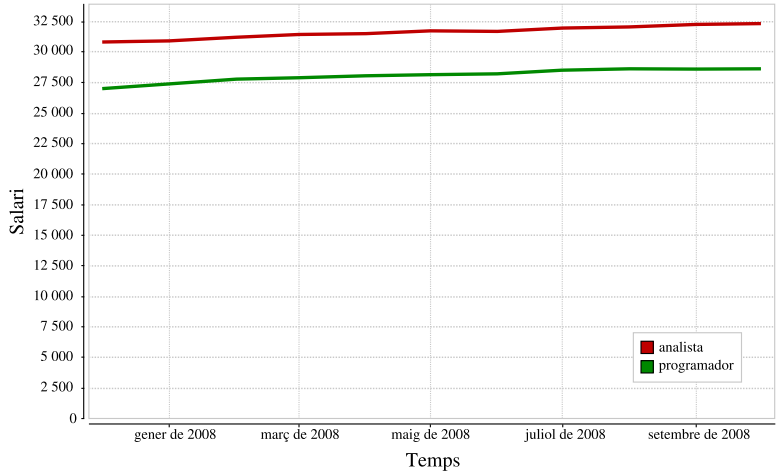


Figura 11.1: Evolució dels salaris mitjos per a un programador i un analista des del gener de 2008 fins al setembre de 2008 [IJT].

Finalment, tal i com veiem a la figura 11.1, assumim que el sou mig d'un programador és de 28 000 €/*any* i el d'un analista és de 31 000 €/*any*. Si suposem que aquestes quantitats és

divideixen en 14 pagues, i que cada mes consta de quatre setmanes amb cinc dies laborables cadascuna d'elles, i amb vuit hores per dia, hem de dividir aquestes quantitats per $14 \times 4 \times 5 \times 8 = 2240$ per a conèixer el sou per hora:

Treballador	**Hores**	**Salari (€/h)**	**Total**
Analista	555	13.84	7 680
Programador	760	12.50	9 500
		Total (€)	17 180

Taula 11.3: Cost en salaris del projecte.

11.2 Cost del maquinari

Una altra part important del projecte és el cost de la infraestructura física que es munta. És necessari disposar de projectors, adquirir les càmeres, etc. La següent taula mostra el cost de cadascun d'aquests components i el total que representa per al projecte:

Concepte	**Unitats**	**Preu unitat (€)**	**Preu (€)**
Portàtil	2	1 200	2 400
Projector	2	800	1 600
Wiimote®	1		60
Càmera Logitech	2	70	140
Prototipus	2		
Connector elèctric	2	2.35	(4.70)
Planxa de metacrilat d'$1m^2$	1	ND	(-)
Guia $1m$	1	ND	(-)
Cargols, arandeles, etc.	-	-	(7.00)
		Cost total	4 210

Taula 11.4: Cost dels components necessaris per a muntar els prototipus.

11.3 Cost total

Finalment, després d'haver analitzat detalladament les principals "despeses" que comporta el projecte, ja és possible calcular-ne el *cost total*:

$$17\ 180\ € + 4\ 210\ € = \mathbf{21\ 390\ €}$$

Capítol 12

Feina futura

Com ja hem dit a la introducció, aquest PFC és la base d'un projecte més ambiciós. En aquest capítol detallarem quins passos i ampliacions caldria fer per completar tot el treball, així com les possibles millores al codi actual.

12.1 Millora del codi actual

Un dels punts febles del codi actual és clarament el calibratge geomètric. El codi ara mateix no aconsegueix un calibratge geomètric amb precisió a nivell de pixel o inclús de subpixel. Per un producte comercial no s'admetria una precisió inferior. Un altre punt necessari seria adaptar i modificar la xarxa per incloure-hi seguretat. Per aquest projecte no era necessari, però si volem que s'utilitzi el codi fora d'aquest context, serà imprescindible.

12.2 Possibles ampliacions del projecte

La primera ampliació que es podria fer seria implementar l'estereovisió. Caldria decidir si es fa activa o passiva, i s'hauria de tenir en compte la construcció de la cave definitiva. Si volem projectar sobre parets qualsevol haurà de ser activa, sinó podrà ser passiva per abaratir costos. Recomanem fer-la activa, per no dependre d'on es projecte.

El següent pas seria implementar un calibratge que corregís la deformació que es produeix al projectar sobre parets, com a la figura 12.1. Aquí ja ens limitem a fer una aplicació on la perspectiva és depenent de l'espectador, perquè la correcció de la deformació es fa en funció del punt de vista de l'usuari. No és cap problema si l'objectiu és fer una *cave flexible*, però no es pot fer si el que volem és una *power wall*.

Una ampliació potser no tant notable, però també important és la correcció cromàtica. Cada projector és lleugerament diferent dels altres, degut als errors de precisió al muntatge i construcció d'aquests. Aquests petits errors provoquen que els colors que projecta difereixin dels altres, per molt que siguin de la mateixa marca i model. També resulta que les parets on es fan les projeccions tampoc són perfectes; poden estar brutes, tenir desnivells, tenir zones més reflectants, etc. Si volem que la *cave flexible* sigui realment flexible, hem de preveure que no sempre les parets seran adequades per projectar-hi, i ho haurem de solucionar. Per resoldre aquest problema, podem implementar alguns algoritmes descrits a [BIWG08] que donen resultats tan espectaculars com els següents:

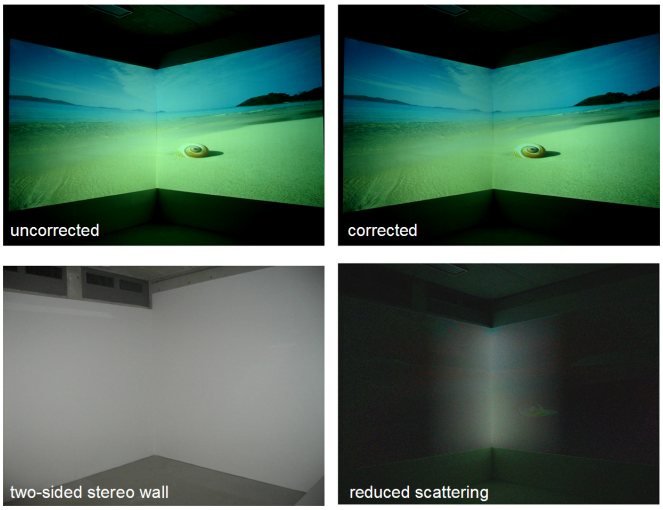


Figura 12.1: Correcció amb més parets.

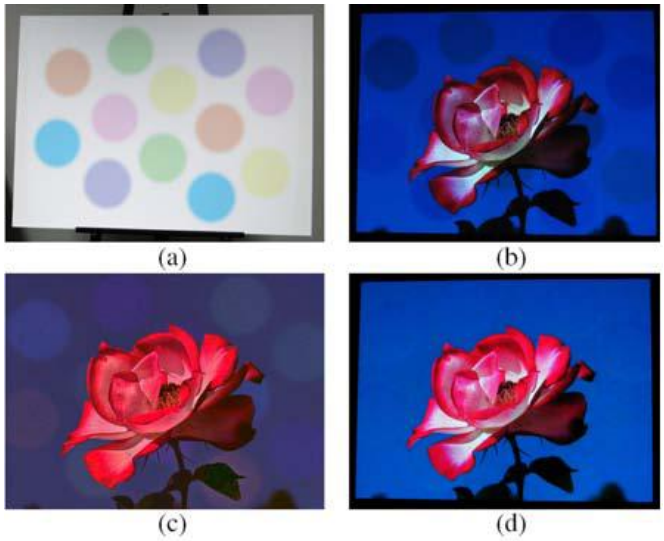


Figura 12.2: Correcció cromàtica. a) Paret amb taques. b) Imatge projectada sense correcció cromàtica. c) Imatge corregida. d) Projecció corregida resultant.

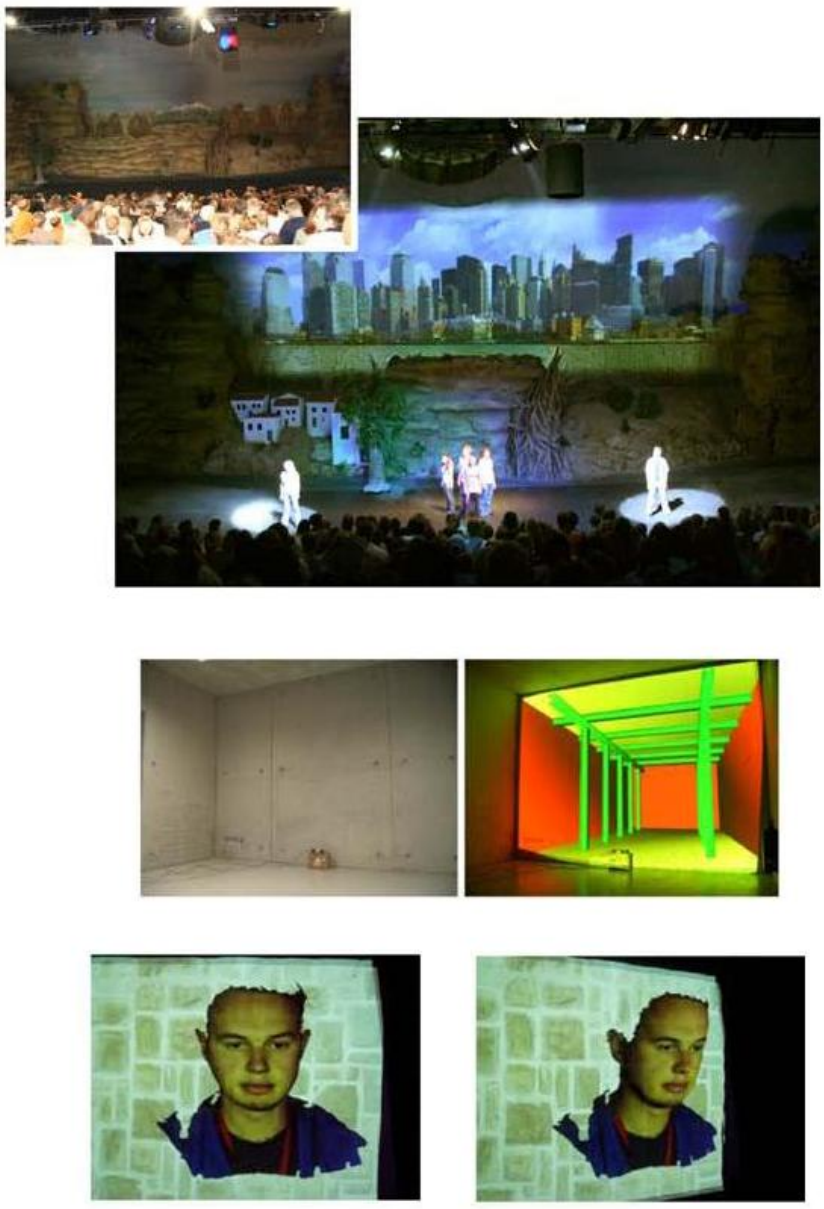


Figura 12.3: Exemples de correccions cromàtiques.A dalt, projecció sobre el decorat en una actuació en viu en el festival de Karl May. Al mig, projecció en una paret de formigó. A sota projecció en una paret de pedra natural

Capítol 13

Conclusions

13.1 Personal

El projecte de final de carrera és l'activitat final d'una etapa universitària i una excel·lent oportunitat per demostrar el coneixement i habilitats obtingudes. Com a estudiants, podem escollir el tema en el qual hi treballarem durant mesos, i, tot i que en informàtica n'hi ha pràcticament una infinitat, les meves preferències em van decantar cap a l'àrea de gràfics i realitat virtual.

13.2 Futur comercial

L'empresa i2cat es dedica a desenvolupar i investigar en l'àmbit d'Internet. Aquesta companyia està interessada comercialment en el nostre PFC per aplicar-ho en un projecte de col·laboració cultural entre diferents regions de Catalunya anomenat Anella Cultural. Les dues parts hem acordat esperar fins acabar la defensa per continuar negociant.

13.3 Agraïments

Per damunt de tot, haig d'agrair-li al meu company, el David Aguilera, la seva paciència i constància en el projecte. En les situacions en les que semblava que no podíem continuar, el seu esperit sempre m'ajudava a progressar i trobar les solucions. També volem agrair els esforços dipositats en nosaltres al nostre tutor, el Pere Brunet, i a un treballador del Centre de Realitat Virtual, el Jordi Moyés. Menció especial mereix l'ajuda que em van prestar dos ex-alumnes de la FIB, el Iago Gayoso i el Kamil Fazil, sense l'ajuda imprescindible dels quals el desenvolupament de l'aplicació 3D hagués estat exponencialment més difícil. Per últim, m'agradaria mencionar també a Margarita Cudolè, que ens va ajudar a veure que hi ha camins que no duen enlloc.

Apèndix A

JMF i *DirectShow*

En aquest annex explicarem què son el JMF i el *DirectShow*, i perquè vam decantar-nos per *DirectShow*.

A.1 JMF

El JMF o *Java Media Framework* és una extensió de Java que permet la programació de tasques multimèdia en aquest llenguatge de programació.

Les principals característiques son:

Estabilitat deguda a que funciona sobre la màquina virtual java (JVM).

Senzillesa, ja que permet, utilitzant poques instruccions, realitzar complexes tasques multimèdia.

Potència, permetent la manipulació d'elements multimèdia de vídeo i àudio locals (procedents de la mateixa màquina a la que s'executa el programa), així como la retransmissió en temps real de vídeo i àudio a través de la xarxa mitjançant el protocol RTP.

JMF no s'inclou a la JDK, ni a la JRE, sinó que s'ha d'instal·lar como un paquet extern.

A.2 *DirectShow*

DirectShow és un entorn multimèdia i una API feta a Microsoft per desenvolupadors de programari que volen operar amb arxius multimèdia o *streams*. Basat en l'entorn *Microsoft Windows Component Object Model* (COM), DirectShow proporciona una interfície comú per operar amb multimèdia en nombrosos llenguatges de programació. És extensible, basat en filtres i pot renderitzar o gravar arxius multimèdia sota demanda. Actualment l'entorn de desenvolupament de *DirectShow* ve integrat amb l'SDK de Windows, i tenim disponibles connectors per poder desenvolupar en java.

A.3 Quin escollir?

Després de veure les característiques de cadascun, com escollir? Hi ha una característica absolutament necessària per realitzar el calibratge, la resolució de les imatges capturades. La màxima resolució de les càmeres de les que disposem és 1600 per 1200, i la volem aprofitar tota.

El JMF va ser desenvolupat en una època en què les *webcams* no tenien una resolució tan bona, i la màxima resolució que admet el JMF és 640 per 480. Així doncs, *DirectShow* és la llibreria que fem servir per captar imatges de la càmera. Obrim un streaming de vídeo en un *buffer* circular, i en el moment que volem una captura, agafem les dades del *buffer*. Això de vegades porta problemes, perquè no podem saber quina imatge del *buffer* estem recollint, per tant, si fem una captura ens hem d'assegurar que tot el *buffer* conté la imatge que volem.

A.4 Enllaços

DirectShow: http://msdn.microsoft.com/en-us/library/ms783323(VS.85).aspx
DSJ: http://www.humatic.de/htools/dsj.htm

Apèndix B

WiimoteLib

Aquesta llibreria implementa la connexió bluetooth amb el Wiimote de forma transparent a l'usuari. També ens permet programar els events provocats pel *Wiimote* per mitjà de *callbacks*. Està implementada en C# i es comunica amb el driver Bluetooth de l'ordinador. El driver que millor funciona i millor s'adapta per la WiimoteLib és el BlueSoleil.

A l'enllaç de la WiimoteLib podrem trobar documentació sobre la mateixa i també una llista de vídeos amb exemples d'aplicacions que s'han fet utilitzant la llibreria.

B.1 Enllaços

BlueSoleil: http://www.bluesoleil.com/

WiimoteLib http://www.codeplex.com/WiimoteLib

Apèndix C

DirectX

En aquest apartat parlarem de què és el *DirectX* i perquè el fem servir en el nostre projecte.

C.1 Què és el *DirectX*?

DirectX es una col·lecció d'APIs creades per facilitar las complexes tasques relacionades amb multimèdia, especialment programació de jocs i vídeo a la plataforma Microsoft Windows.

Consta de les següents APIs:

Direct3D : utilitzat pel processat i la programació de gràfics en tres dimensions (una de les característiques més utilitzades de DirectX).

Direct Graphics : per dibuixar imatges en dos dimensions (planes), i per representació d'imatges en tres dimensions.

DirectInput : utilitzat per processar dades del teclat, mouse, joystick i altres controls pels jocs.

DirectPlay : per comunicacions en xarxa.

DirectSound : per la reproducció i gravació de sons d'ones.

DirectMusic : per la reproducció de pistes musicals compostes amb DirectMusic Producer.

DirectShow : per reproduir àudio i vídeo amb transparència de xarxa.

DirectSetup : per la instal·lació de components DirectX.

Tot i ser desenvolupat exclusivament per la plataforma Windows, una implementació d'aquesta API es troba en progrés per sistemes Unix (en particular Linux) i X Window System coneguda com Cedega, desenvolupada per l'empresa de programari Transgaming i orientada a l'execució de jocs desenvolupats per Windows a sistemes Unix.

La decisió d'utilitzar-la és deguda a que hem reaprofitat un codi ja existent que l'utilitzava.

C.2 Per què el fem servir?

Tota la part gràfica de l'aplicació del nostre PFC està implementada amb *DirectX*. Està molt ben integrada tant amb el C# com amb HLSL, el llenguatge que hem fet servir per programar els shaders.

Apèndix D

Shaders

Un *shader* és un programa senzill capaç de manipular dades de coordenades en 3D i textures. Aquests programes s'executen a la GPU de la targeta gràfica, i així, al tenir tota la informació a la memòria pròpia de la targeta, evitem col·lapsar el bus que ve de la memòria principal, i, per tant, es pot manipular la sortida gràfica d'una manera considerablement més ràpida que amb un programa executat al processador. Qualsevol aplicació que faci un ús intensiu en gràfics necessita desenvolupar *shaders*.

Els *shaders* que hem programat fan una feina ben senzilla: els utilitzem per corregir geomètrica i cromàticament la imatge resultant. Desenvolupar aquests, però va ser desafiant, perquè mai n'havíem fet cap.

Per començar a dissenyar aquesta part, la primera gran decisió era trobar el llenguatge de *shading* més adequat. Ens vam decantar per HLSL perquè s'integra perfectament amb DirectX, i perquè la documentació i exemples que vam trobar eren molt bons. Després de llegir innumerables tutorials per aprendre'n, i després de fer innumerables proves, ens vam trobar amb alguns problemes per fer un shader que corregís les coordenades d'un punt 3D a partir d'una matriu 3x3. Expliquem primer què és HLSL i continuem pels problemes que ens ha donat.

D.1 HLSL

El *High Level Shader Language o High Level Shading Language* (HLSL) és un llenguatge de *shading* propietari desenvolupat per Microsoft per utilitzar la API *Microsoft Direct3D*. És anàleg al GLSL que fa servir OpenGL i és molt similar al *Cg shading language* de NVIDIA

Els programes HLSL es reparteixen en tres formes: *vertex shaders, geometry shaders, and pixel (o fragment) shaders*. Un *vertex shader* és executat per cada vèrtex que s'envia a l'aplicació, i és el responsable de transformar les coordenades d'espai a coordenades de vista, generar coordenades de textura i calcular els coeficients de llum. Quan un grup de vèrtexs (normalment 3, que formen un triangle) passen al *vertex shader*, les seves coordenades normalment acaben omplint un àrea de piexls, en un procés que en coneix com rasterització. Cada un d'aquests pixels passa pel *pixel shader*, on es calcula el color que es veurà a la pantalla.

D.2 Aplicar les Homografies

Ja hem vist que una homografia permet transformar "una imatge en una altra". L'aplicació d'usuari, però, té un model 3D, no una imatge. Tot i que seria possible dibuixar l'escena en un *frame* i aleshores aplicar-li la homografia, és molt millor fer-ho dins del procés de transformació

del model tridimensional a imatge. El principal problema que es té és la sintaxi, perquè s'haurà de manipular dades tant en 3D com en 2D.

El resultat de multiplicar un punt del model per les matrius de modelat i projecció és un punt amb tres coordenades més l'homogènia. Per a fer la correcció geomètrica es volen coordenades 2D i, per tant, o bé es crea un vector 2D homogeni per poder-lo multiplicar per la matriu 3x3, o bé es manipula la matriu per convertir-la en una 4x4. Per raons de sintaxi en HLSL (*High Level Shader Language*, Llenguatge de *shading* propietat de Microsoft)[1], la segona opció és molt més fàcil d'implementar i també més eficient.

Per a aplicar una homografia en 2D (coordenades homogènies) es fa de la següent manera:

$$\begin{pmatrix} x_q \\ y_q \\ r \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix}$$

Per tenir una equivalència en 3D que conservi la coordenada z, es pot manipular l'homografia i afegir-li una dimensió. Només es vol modificar x, y, conservant la z. Afegint una columna a 0 a la tercera posició s'aconsegueix el primer objectiu, perquè s'anul·la la component z. Si s'intercala una fila a la tercera posició, amb un 1 a la tercera columna i la resta zeros, s'obté la z sense modificar.

$$\begin{pmatrix} x_q \\ y_q \\ z_q \\ r \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & 0 & h_{13} \\ h_{21} & h_{22} & 0 & h_{23} \\ 0 & 0 & 1 & 0 \\ h_{31} & h_{32} & 0 & h_{33} \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$$

Aquest resultat, però, no es pot passar tal qual al *pixel shader*, perquè abans d'arribar-li el *pipeline* agafa la component homogènia i divideix tot el vector per ella, de tal manera que no tindríem z_q, sinó z_q/r. L'únic que caldrà fer abans d'acabar és dividir x_q i y_q per r i posar la component homogènia a 1. Així doncs, el punt resultant és (x_q/r, y_q/r, z_q, 1).

[1] Per a aplicar les transformacions d'una homografia s'han utilitzat *shaders*.

Apèndix E

Visió per computador

La visió per computador és la ciència de les màquines capaces d'utilitzar i manipular dades visuals. Aquestes dades poden ser de diferents tipus, com vídeo, imatges estàtiques, o inclús dades tridimensionals, com succeeix al camp de la medicina. En el nostre cas, tractem sempre amb imatges estàtiques.

E.1 Llibreries utilitzades

La llibreria que hem utilitzat per fer les captures ha estat DirectShow, explicada a l'apèndix A, i la que hem fet servir per fer operacions típiques de visió per computador ha estat Javavis. Aquesta última, és una llibreria feta pel Departament d'Informàtica i Intel·ligència Artificial de la Universitat d'Alacant i està desenvolupada en java. Les raons per escollir-la han estat que agrupava totes les operacions de visió que necessitàvem a priori i que el codi és gratis i lliure, fet que ens ha facilitat feina alguns cops on necessitàvem alguna funcionalitat més de les que ja ofereix.

E.2 Algoritmes comuns

Els algoritmes més comuns per imatges estàtiques serveixen sobretot per descartar informació indesitjada i per eliminar soroll degut a defectes de la càmera, un mal enfoc o canvis en la il·luminació. Per eliminar els efectes d'una eliminació defectuosa, se sol equalitzar la imatge (o també en diem rehistogramar), que vol dir que calculem en quin rang de colors està tota la informació i la redistribuïm. És a dir, suposem que la càmera ens dona informació en escala de grisos de 0 a 255, i degut a que hi ha massa llum, totes les dades ens arriben amb valors de 100 a 255. Agafem aquestes dades i les redistribuïm de manera que ocupin tot el rang de 0 a 255, i així augmentem el contrast, cosa que ens ajudarà a fixar valors per binaritzar. Binaritzar vol dir passar una imatge amb una escala de valors gran a una imatge de només blancs purs i negres purs. La manera més usual és fixar uns llindars amb la imatge d'entrada a partir dels quals considerem que el color és blanc, i la resta negre. Per exemple, si fixem com a valor de binarització el 100, el color 123 serà substituït per 1 i el color 76 per un 0. A la figura E.2 podem veure un exemple de binarització i a la figura E.1 una d'equalització.

Figura E.1: Exemple d'una rehistogramació. El resultat és la imatge de la dreta

Figura E.2: Exemple d'una binarització.

E.3 Enllaços

Javavis: http://javavis.sourceforge.net/

Apèndix F

Instal·lació i configuració

Aquest annex descriu els passos que s'han de fer per tal de poder executar el nostre projecte, descrivint com instal·lar tot el que necessita l'ordinador i com configurar-lo.

Se suposa que el sistema operatiu de què disposem és un *Microsoft® Windows® XP* correctament actualitzat (Service Pack 3).

F.1 Arbre de directoris del CD

L'arbre de directoris del CD és el següent:

```
cdrom
|-- misc
|-- pfc
|   |-- doc
|   |-- part1
|   |   |-- bin
|   |   |-- lib
|   |   `-- src
|   `-- part2
`-- software
    `-- installers
```

- El directori `misc` conté vídeos d'exemple.
- El directori `pfc` conté aquesta memòria i el seu codi font, així com el codi font i compilat del mòdul de calibratge i de l'aplicació d'usuari.
- El programari addicional que cal instal·lar està correctament ubicat dins del directori `software/installers`.

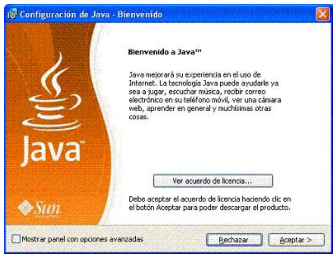
F.2 Instal·lació de Java

Abans d'executar l'instal·lador de Java és necessari que tanqueu totes les aplicacions, fins i tot el navegador web.

1. Feu doble clic a l'instal·lador `jre-6u3-rc-windows-i586.exe`.

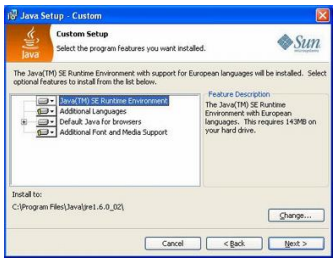
   L'instal·lador descomprimirà tots els arxius necessaris per a la instal·lació.

2. A la finestra *Configuració de Java - Benvingut*, l'instal·lador us permet veure el *Contracte de llicència*. Accepteu-lo fent clic al botó *Accepta* per continuar el procés d'instal·lació.



3. Tot seguit es mostren diferents elements que es poden instal·lar, com la *Barra de Google per a Internet Explorer* i *Google Desktop*. Escolliu el que vulgueu i feu clic a *Següent*.

4. L'instal·lador mostra una nova pantalla d'*instal·lació personalitzada*. Deixeu les opcions per defecte i feu clic a *Següent*.



5. Apareix un diàleg indicant quin és el progrés de la instal·lació.

6. Tot seguit, s'obriran diversos diàlegs per a realitzar les darreres etapes de la instal·lació. Finalment, apareixerà un missatge de confirmació, agraint-vos haver instal·lat Java.

F.3 Instal·lació de *DirectShow Java wrapper*

Descomprimiu l'arxiu `dsj.zip` al vostre escriptori i realitzeu els següents passos:

1. Assegureu-vos de tenir instal·lat *DirectX 9* i el reproductor *Windows Media Player9 9* o superior. Si no els teniu instal·lats no disposareu de tots els descodificadors necessaris i és possible que els dispositius de captura no funcionin.

2. Poseu el fitxer `dsj.dll` al directori `jre/bin` que trobareu al directori d'instal·lació de Java (generalment, `C:\Arxius de programa\Java`), o al directori de Sistema (`C:\WINDOWS\System32`).

3. Poseu el fitxer `dsj.jar` al direcotri `jre/lib/ext`.

Si després d'executar aquests passos el DSJ encara no funciona, comproveu si disposeu de més d'una instal·lació de Java. En tal cas, assegureu-vos d'haver copiat els fitxers a la ubicació de la *Java Runtime Environment* que s'està executant.

F.4 Instal·lació del *driver Blue Soleil*

1. Feu doble clic al fitxer `blue-soleil-setup.exe` per tal d'iniciar l'instal·lador.
2. Apareix una finestra on podeu seleccionar l'idioma de l'instal·lador. Seleccioneu-ne un i feu clic a *Següent*.
3. Seguiu les instruccions en pantalla.
4. Torneu a iniciar el vostre ordinador.

F.5 Configuració de la xarxa

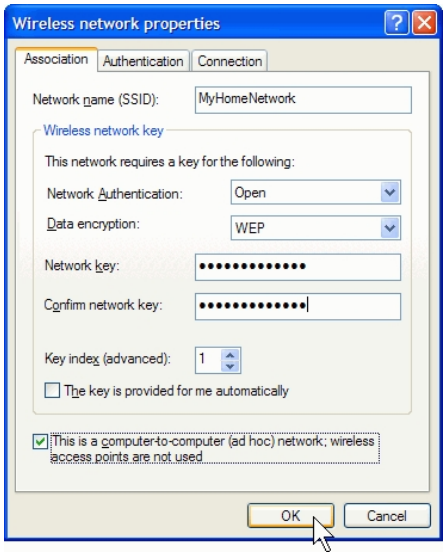
A les xarxes amb *routers* sense fils, aquests s'encarreguen de coordinar les comunicacions entre els ordinadors que hi estan connectats. A les xarxes *ad hoc* cal designar un ordinador per a què actuï com a tal.

F.5.1 Configurar el primer ordinador

1. Feu clic a *Inici* i, a continuació, a *Panell de control*.
2. A la finestra d'*Escolliu una categoria*, seleccioneu *Treball en xarxa i connexions a Internet*.
3. Feu clic amb el botó dret a la vostra connexió a la xarxa sense fils i seleccioneu *Propietats*.

4. Al diàleg *Propietats de la connexió a la xarxa sense fils*, feu clic a la pestanya *Xarxes sense fils*.
5. Dins la pestanya *Xarxes sense fils*, a la zona de *Xarxes preferides*, feu clic a *Afegeix*.
6. A la finestra *Propietats de la xarxa sense fils*, a la pestanya anomenada *Associació*, escriviu el no de la vostra xarxa *ad hoc* al camp *Nom de la xarxa (SSID)*.
7. Deixeu sense marcar la casella *La clau és proporcionada per mi automàticament* i seleccioneu *Això és una xarxa d'ordinador - a - ordinador (*ad hoc*)*.

8. Creeu una contrasenya de 13 dígits i escriviu-la als camps *Contrasenya de la xarxa* i *Confirmeu la contrasenya de la xarxa*. Per a augmentar la seguretat, utilitzeu lletres, nombres i símbols de puntuació. Finalment, feu clic a *D'acord*.



9. Feu novament clic a *D'acord* per a desar els canvis.

F.5.2 Configurar ordinadors addicionals

1. Feu clic amb el botó dret a la icona *Xarxes sense fils* que hi ha a la part inferior dreta de la vostra pantalla, i llavors seleccioneu *Mostra les xarxes sense fils disponibles*.

2. La finestra de *Connexió a xarxa sense fils* apareix i mostra la xarxa sense fils que heu configurat prèviament amb el nom (SSID) que haguéssiu escollit. Si no la veieu, feu clic a *Refresca la llista de xarxes*, a l'extrem superior esquerre. Feu clic a la vostra xarxa i, tot seguit, al botó *Connecta* de la cantonada inferior dreta.

3. El sistema us demana una contrasenya. Escriviu als dos camps la contrasenya que abans heu creat i feu clic a *Connecta*.

Un cop connectats, podeu tancar la finestra *Connexió a xarxa sense fils*.
Repetiu aquests tres passos a cada ordinador que vulgueu connectar a la xarxa *ad hoc*.

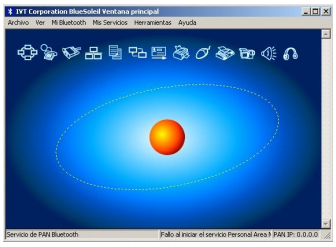
F.6 Sincronitzar un *Wiimote®*

El primer que s'ha de fer per a sincronitzar un *Wiimote®* amb l'ordinador és iniciar el programa *BlueSoleilImatges obtingudes a [CP].*:

1. Feu clic amb el botó dret del ratolí a la icona de *Bluetooth* que trobareu a la safata de sistema de la barra d'inici (part inferior dreta de la pantalla).

2. Seleccioneu l'opció *Mostra vista clàssica*.

3. Una finestra com la següent apareix:



Un cop iniciat el programa, podeu agafar el comandament i sincronitzar-lo:

1. Traieu la tapa de les piles del *Wiimote*® i premeu el botó vermell.

2. Mentre les llums del *Wiimote*® parpellegin, feu doble clic a l'esfera taronja de la finestra principal de *BlueSoleil*.

3. El sistema ha reconegut tots els dispositius *Bluetooth*. En concret, el *Nintendo RVL-CNT-01*, que és el *Wiimote*®.

4. Tornem a prémer el botó vermell del comandament.

5. Amb el botó dret del ratolí, feu clic sobre el dibuix del *Wiimote*® i escolliu l'opció *Connecta* i, tot seguit, *Servei de dispositiu d'interfície humana de* Bluetooth.

6. Després d'això, el comandament ja està sincronitzat i podem posar-hi novament la tapa. El resultat és el que veiem a continuació.

F.7 Instal·lació i configuració de l'aplicació d'usuari

Per tal d'instal·lar l'aplicació d'usuari només cal que copieu el contingut del CD ubicat a `pfc/part2` a algun directori del vostre sistema.

Tot seguit, heu d'editar l'arxiu de configuració `common-config.txt`[1], de tal manera que les ubicacions que s'hi especifiquen puguin ser escrites pel mòdul de calibratge i llegides per l'aplicació d'usuari.

F.8 Instal·lació i configuració del mòdul de calibratge

Per tal d'instal·lar el mòdul de calibratge només cal que copieu els continguts del directori `pfc/part1/bin`. Això inclou els arxius `master.bat`, `lworker.bat` i `rworker.bat`, que serveixen per a iniciar el programa, així com `pfc-calibrator.jar` i `default-configuration.xml`.

El fitxer que haureu d'editar per tal de què el mòdul de calibratge es vinculi correctament a l'aplicació d'usuari és `default-configuration.xml`. Aquest conté una configuració per defecte del mòdul que funciona correctament, però caldrà que editeu el següent camp

```
<param name="common-config-file" value="C:\dp\common-config.txt" />
```

per a què el fitxer apunti al que hi ha a l'arrel de l'aplicació d'usuari.

[1] Vegeu l'annex **??**.

Apèndix G

Manual d'usuari

En aquest annex s'explica com utilitzar el sistema: per una banda, s'ensenyen els passos que cal seguir per a calibrar-lo i fer que els resultats siguin accessibles per l'aplicació d'usuari; per l'altra, es comenten les principals opcions que es poden activar de l'aplicació d'usuari.

Recordeu que tots els mòduls han d'estar connectats a la mateixa xarxa sense fils, tal i com s'explica a l'annex F.5.

G.1 Mòdul de calibratge

Per a utilitzar el mòdul de calibratge és necessari que inicieu, primer, el *master* i, tot seguit, hi connecteu tots els *workers*. Quan el sistema tingui registrats tots els mòduls, podreu començar amb el procés de calibratge.

G.1.1 Iniciar el *master*

Per tal d'iniciar el *master*, obriu un terminal:

1. Feu clic a *Inici* i, tot seguit, escolliu *Executa*.
2. Al camp *Obra*, escriviu `cmd`.
3. S'obra una finestra amb un terminal.

Tot seguit, heu d'accedir al directori on estigui instal·lat el mòdul de calibratge i executar la següent comanda:

```
C:\pfc\calibratge\> master.bat
```

D'aquesta manera, el sistema ens mostrarà el següent menú:

```
AVAILABLE OPTIONS
--Misc options--
0.- Quits
1.- Print available options
2.- Print number of registered workers

--Load/save data--
3.- Load new configuration file
4.- Load a previously computed geoalign result
5.- Load previously computed chromatic masks
```

```
6.- Ask all clients to save their data (homographies, masks, etc.)

--Geometric alignment tasks--
7.- Compute darkener factors for each client
8.- Geometric alignment
9.- Fix final display manually

--Chromatic alignment tasks--
10.- Chromatic alignment

--Testing tasks--
11.- Send an image to the projectors (geo/chrom testing)
12.- Paint projectors' contours
13.- (Dummy task)
======================================================================

>> Enter an option:
```

Conèixer la IP del *master*

Per tal de poder iniciar els *workers* necessitareu conèixer l'adreça IP del *master*, ja que la necessiten per connectar-s'hi. Executeu la següent comanda en un terminal i fixeu-vos en el valor de la línia *Adreça IP*:

```
C:\> ipconfig
```

G.1.2 Iniciar un *worker*

Per tal d'iniciar un *worker* obriu un terminal i dirigiu-vos al directori on heu instal·lat el mòdul de calibratge (tal i com es descriu anteriorment). Aleshores, executeu la comanda

```
C:\pfc\calibratge\> lworker
```

si el *master* i el *worker* són al mateix ordinador, o

```
C:\pfc\calibratge\> rworker 192.168.1.1
```

si estan en ordinadors diferents. En el segon cas, heu de canviar `192.168.1.1` per l'adreça IP que tingui realment el *master*.

G.1.3 Modificar la configuració

A l'apartat d'instal·lació s'explica com modificar el fitxer `default-configuration.xml` per tal de poder utilitzar l'aplicació. El primer que cal fer quan s'hagi iniciat el *master* i hi hagi tots els *workers* connectats és carregar aquesta nova configuració:

1. Seleccioneu l'opció 3, `Load new configuration file`.
2. Indiqueu la ubicació del nou fitxer.

G.1.4 Realitzar un calibratge geomètric

1. Seleccioneu l'opció 8, `Geometric alignment`.

2. Les pantalles de tots els ordinadors es posaran en negre i la projecció de patrons s'iniciarà. Cada vegada que un mòdul acabi de projectar el seu patró, tots els mòduls intentaran calcular les respectives homografies.

3. Quan el calibratge acabi, el sistema us preguntarà on voleu desar el resultats obtinguts (per si voleu recuperar-los més endavant). Indiqueu-li una ubicació que existeixi i un nom de fitxer qualsevol, com per exemple:

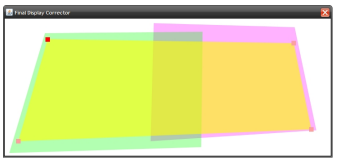
```
> Enter a filename: C:\tmp\geoalign.data
```

G.1.5 Corregir la pantalla final

Després d'haver dut a terme un calibratge geomètric *satisfactori*:

1. Seleccioneu l'opció 9, `Fix final display manually`.

2. El sistema projecta la pantalla final que hi ha calculada en aquell moment pels projectors, per tal de què pugueu veure com queda. El *master* també mostra una finestra com la següent, que permet modificar-ne les quatre cantonades.



3. Quan el resultat que veieu projectat sigui satisfactori, tanqueu la finestra per a donar per finalitzada la correcció manual.

4. El sistema us demanarà que indiqueu on desar els resultats. Indiqueu-li una ubicació que existeixi i un nom de fitxer qualsevol, com per exemple:

```
> Enter a filename: C:\tmp\geoalign-ok.data
```

G.1.6 Realitzar un calibratge cromàtic

Després d'haver dut a terme un calibratge geomètric *satisfactori*:

1. Seleccioneu l'opció 10, `Chromatic alignment`.

2. El sistema comença el procés de correcció cromàtica. En funció dels paràmetres de configuració, és possible que projecti o no algun patró.

3. Quan el sistema acabi de realitzar els càlculs us demanarà que indiqueu on desar els resultats. Indiqueu-li una ubicació que existeixi i un nom de fitxer qualsevol, com per exemple:

```
> Enter a filename: C:\tmp\chrom.data
```

G.1.7 Obrir resultats

Tal i com s'explica als passos anteriors, després de dur a terme cadascuna de les etapes de calibratge és possible desar-ne els resultats. El sistema ens permet recuperar-los en qualsevol moment:

- Per tal de recuperar una *alineació geomètrica*, seleccioneu l'opció 4, `Load a previously computed geoalign result`, i indiqueu el nom del fitxer.
- Per tal de recuperar una *correcció cromàtica*, seleccioneu l'opció 5, `Load previously computed chromatic masks`, i indiqueu el nom del fitxer.

G.1.8 Desar els resultats d'intercanvi

Per tal de desar els resultats obtinguts durant els processos de calibratge en un format que l'aplicació d'usuari entengui, i a les ubicacions especificades pel fitxer `common-config.txt`, escolliu l'opció 6, `Ask all clients to save their data (homographies, masks, etc.)`.

G.1.9 Sortir de l'aplicació

Per tal de sortir de l'aplicació, seleccioneu l'opció 0, `Quit`.

G.2 Aplicació d'usuari

En aquesta secció s'explica amb cert detall les funcionalitats que s'han afegit a l'aplicació [CL]. Les que s'han heretat, en canvi, es mostren breument en una llista.

G.2.1 Habilitar l'ús de les homografies

Cada mòdul disposa de la descripció total del món virtual. Això vol dir que poden dibuixar l'escena individualment i mostrar-la correctament.

Si es vol veure l'escena a través dels projectors, cal habilitar les homografies amb la tecla `[E]` per tal de què el que mostra cadascun d'ells sigui la porció de món que li pertoca.

G.2.2 Habilitar l'ús de les màscares

Per a poder aplicar les màscares cromàtiques s'ha de prémer la tecla `[M]`.

G.2.3 Reiniciar la posició central

Tot i que aquesta funcionalitat ja estava present a [CL], s'explica apart degut a la seva importància. Quan l'usuari s'ha situat a una posició que ell considera "natural" i que es correspon al "centre" del món virtual (mirant, òbviament, a la pantalla), ha de prémer la tecla *Espai* per a què la càmera virtual se situï a la posició inicial.

G.2.4 Altres funcionalitats

El sistema té altres funcionalitats que es poden activar i desactivar:

`[G]` Mostra o amaga la graella.

`[H]` Mostra o amaga estadístiques del sistema.

[L] Mostra o amaga les línies que van dels objectius a la paret del fons.

[R] Reubica els objectius.

[S] Mostra o amaga estadístiques relacionades amb el *Wiimote*®.

[T] Mostra o amaga els objectius.

Apèndix H

Resultats

Aquest annex mostra els diferents resultats que s'han obtingut tant al mòdul de calibratge com a l'aplicació d'usuari. Les imatges que hi ha pertanyen a execucions reals del sistema.

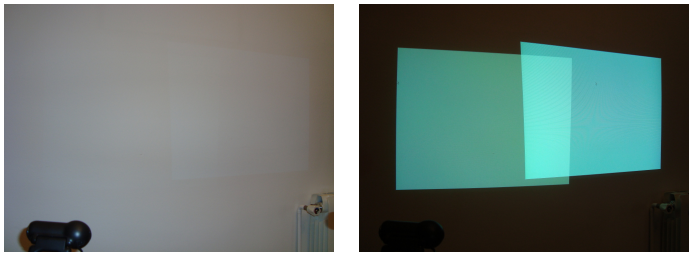
H.1 Mòdul de calibratge

A continuació ens centrarem en els resultats que dóna el mòdul de calibratge. En concret, es mostren els diversos passos que realitza l'algorisme amb el resultat que ofereixen cadascun d'ells.

Per tal de veure el procés de calibratge en funcionament, podeu mirar el vídeo que trobareu inclòs al CD adjunt a aquesta memòria.

H.1.1 Els patrons

La manera que té el sistema per tal de conèixer la disposició de les projeccions sobre la paret és enviant patrons i analitzant-los. La figura H.1.1 mostra la paret on es projectaran les àrees (H.1(a)) i la zona que ocupa cadascuna d'elles (H.1(b)).



(a) Paret on es projectarà el món virtual.

(b) Àrees ocupades pels projectors.

Figura H.1: Zona de projecció.

Després de què tots els mòduls hagin capturat el fons, s'inicia l'enviament i captura de patrons[1], com el que podeu veure a la captura de la figura H.1.1.

Quan es disposa d'aquestes captures, s'apliquen els algorismes de visió per computador que permeten obtenir la informació dels patrons de les imatges (vegeu la figura H.1.1.

[1] És important recordar que, de fet, el que s'envia i es captura són *components d'un patró*.

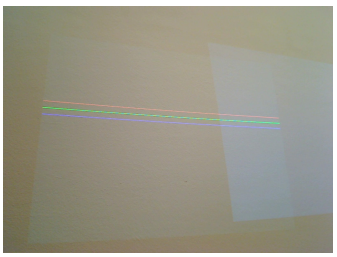
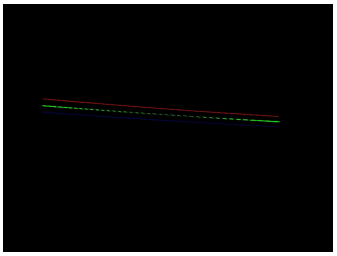


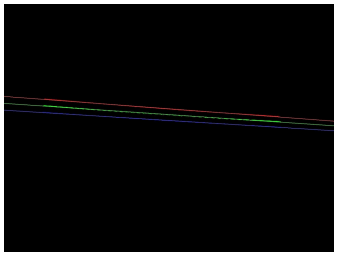
Figura H.2: Patró capturat per un dels mòduls.

(a) Imatge filtrada fent la diferència entre la captura i el fons.

(b) Imatge filtrada aplicant una binarització a H.3(a).



(c) Imatge filtrada on es determina el color de cada línia.



(d) Imatge filtrada on s'hi ha superposat la recta calculada analíticament, després d'aplicar un algorisme de regressió.

Figura H.3: Evolució del processat d'una captura a l'aplicar-li els diversos filtres.

H.1.2 La pantalla final

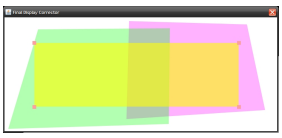
El resultat d'haver processat els patrons és disposar de la informació sobre les àrees projectades, de tal manera que es pugui calcular la pantalla final. La figura H.4 mostra la pantalla final que s'ha calculat automàticament, la qual es veu com a un rectangle des de la càmera del mòdul 1.

Si l'usuari modifica manualment les coordenades d'aquesta pantalla, podrà alinear-la amb el terra i les columnes.

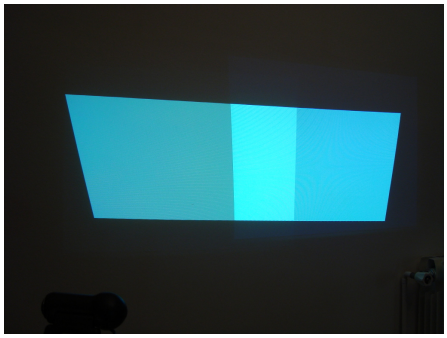
H.1.3 Correcció cromàtica

Finalment, quan ja es té calculada la geometria de la pantalla final, es poden calcular les màscares que faran la correcció cromàtica.

La figura següent mostra els diferents resultats que s'obtenen a l'aplicar les homografies sobre una imatge qualsevol i projectar-la, en funció de la màscara que se l'hi apliqui.

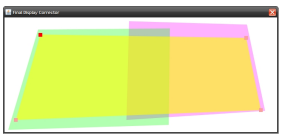


(a)

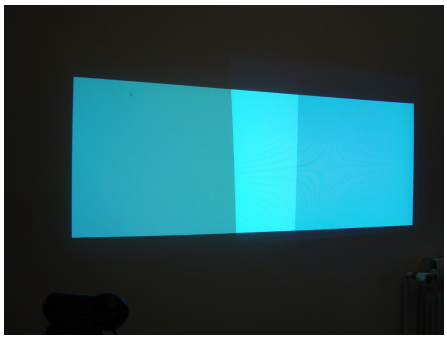


(b)

Figura H.4: Pantalla final calculada automàticament.



(a)



(b)

Figura H.5: Pantalla final corregida manualment per l'usuari.

La figura H.6(b) posa de manifest el problema descrit a la secció **??**. Després de corregir-ho, el resultat final és el de la figura H.6(c).

H.2 Aplicació d'usuari

Tot seguit, es mostren unes quantes captures dels resultats obtinguts a l'aplicació d'usuari. De totes maneres, en aquest cas es recomana veure el vídeo contingut al CD adjunt a la memòria,

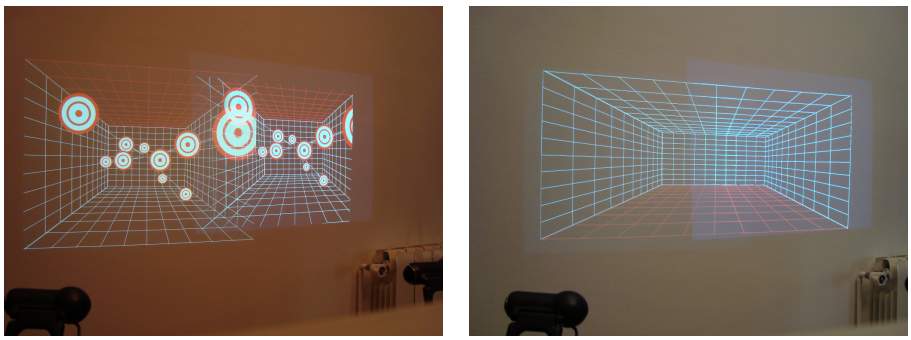
(a) No s'aplica cap màscara. (b) S'aplica la màscara amb degradats. (c) S'aplica la màscara amb degradats corregits.

Figura H.6: Resultats obtinguts amb el procés de calibratge, aplicant diferents màscares per a la correcció cromàtica.

el qual permet comprendre molt millor el funcionament d'aquesta part del projecte.

H.2.1 Homografies

L'aplicació d'usuari permet activar o desactivar l'ús de les homografies. Tal i com es pot apreciar a la figura H.2.1, el sistema només és coherent si estan habilitades, de tal manera allò que es mostra és coherent.

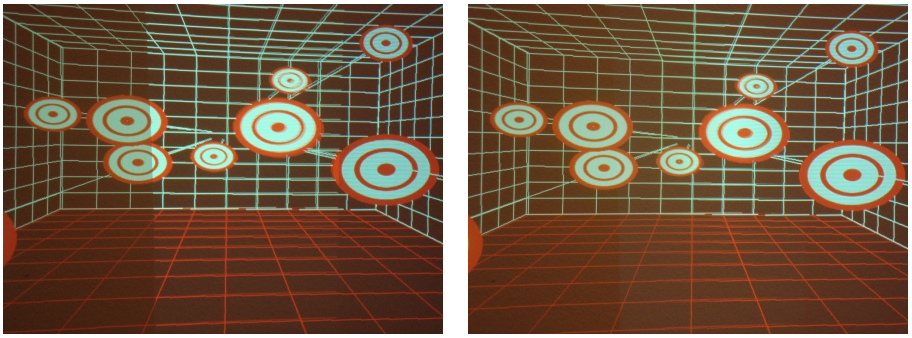


(a) (b)

Figura H.7: Exemple de l'aplicació d'usuari utilitzant o no les homografies.

H.2.2 Màscares

L'aplicació també permet habilitar i deshabilitar les màscares. Utilitzar-les és aconsellable, ja que aconsegueix dissimular els petits errors de calibratge que s'hagin pogut produir. La següent figura mostra ambdós resultats:



(a) (b)

Figura H.8: Exemple de l'aplicació d'usuari amb o sense màscares.

Bibliografia

[BIWG08] Oliver Bimber, Daisuke Iwai, Gordon Wetzstein, and Anselm Grundhöfer. The Visual Computing of Projector-camera Systems. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, pages 1–25, New York, NY, USA, 2008. ACM.

[BV99] Volker Blanz and Thomas Vetter. A Morphable Model for the Synthesis of 3d Faces. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, New York, NY, USA, 1999. ACM.

[CL] Johnny Chung Lee. Head Tracking for Desktop VR Displays using the Wii Remote. http://www.cs.cmu.edu/ johnny/projects/wii/.

[CP] Usar el Wiimote en el PC. http://en.wikipedia.org/wiki/Adapter_pattern.

[CSWL02] Han Chen, Rahul Sukthankar, Grant Wallace, and Kai Li. Scalable Alignment of Large-Format Multi-Projector Displays Using Camera Homography Trees. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 339–346, Washington, DC, USA, 2002. IEEE Computer Society.

[IJT] InfoJobs Trends Salarios. http://salarios.infojobs.net/.